FEBRUARY 1991 £2.75

Commodore DISK·USER·

THE CRANMORE DIAMOND AFFAIR.

THE BASICS OF BASIC

TRIVIA CHALLENGE RESULTS

9 770953 061007

...it's dynamite!

POWER CARTRIDGE

FOR YOUR COMMODORE 64/128

"...unbelievable value for money" ZZAPP! Dec 89

- POWER TOOLKIT
- POWER MONITOR
- TAPE & DISK TURBO
- PRINTERTOOL
- POWER RESET
- TOTAL BACKUP

..."Monay well spent" YC/CDU Jan 90

42 Pg Manual - "Damnad Good Handbook" CCI Jan 90

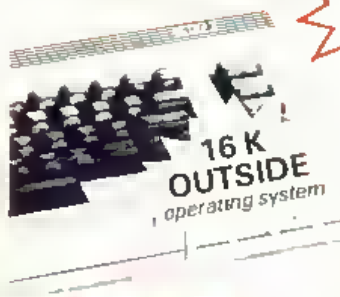
SO MUCH FOR SO LITTLE

AVAILABLE FROM ALL GOOD OOMPUTER RETAILERS

TRIED AND TESTED - OVER 100,000 SOLD IN EUROPE

YOU WILL WONDER HOW YOU EVER MANAGED WITHOUT IT

...."highly recommanded for C64 users" CCI Jan 90



POWER TOOLKIT

A powerful BASIC toolkit providing that incredibly helpful programmers that considerably simplifies programming and debugging.

AUTO	HARDCAT	RENUMBER
AUDIO	HARDCOPY	REPEAT
COEOR	HEX$	SAVE
DEEK	INFO	TRACE
DELETE	KEY	UNNEW
DOKE	PAUSE	QUIT
DUMP	PEEST	MONITOR
FIND	RLOAD	RLOAD

RENUMBER Allows modification of the BASIC line numbers. Allows part of a program to be renumbered or displayed.

PSET Set up printer type

HARDCAT Print out Directory

The toolkit commands can be used in your programs.

TAPE & DISK TURBO

Using POWER CARTRIDGE you can load up to 10 times faster with your data recorder. The tape commands can be used in your own programs.

LOAD SAVE VERIFY

MERGE AUDIO

POWER MONITOR

A powerful machine language monitor that is readily available and leaves all of your Commodore memory available for programming.

Also works in BASIC ROM, KERNAL and I/O area.

A ASSEMBLE	E INTERPREE	S SAVE
E COMPARE	J JUMP	T TRANSFER
D DIS-ASSEMBLE	L LOAD	V VERIFY
	M MEMORY	W WALK
E EDIT	R PRINT	X EXIT
G GO	B REGISTER	$ DIRECTORY
H HUNT		DOS Commands

PRINTERTOOL

The POWER CARTRIDGE contains a very effective printer interface that self detects if a printer is connected to the Serial Bar or User Port.

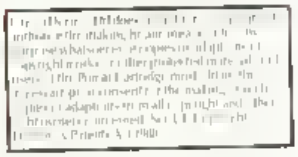
It will print all Commodore characters on Epson and compatible printers.

The printer interface has a variety of set up possibilities. It has a built in HARDCOPY of screens not only for Serial printers (MPS801, 802, 803 and but also for Centronic printers (EPSON, STAR, CITIZEN, PANASONIC and).

The HARDCOPY function automatically distinguishes between HIRES and MULTICOLOUR graphics and converts into shades of grey. The PSET function allows you to decide on Large/Small and Normal/Reverse printing.

The printer PSET functions are:

PSET 0	Self detection Serial/Parallel
PSET 1	EPSON mode only
PSET 2	SEIKOSHA CONSUL mode only
PSET 3	Turns the printer to Graphics
PSET 4	HARDCOPY setting for MPS803/1526
PSET 8	Bit image mode
PSET C	Setting Power Cipher Card sending Control Codes
PSET T	All characters are printed in an unmodified state
PSET U	Runs a Serial printer and leaves the User-port available
PSET Sx	Sets the Secondary address for HARDCOPY with Serial Bus
RSET LF	Adds a line feed (LINES) after every line
PSET F0	Switches PSET F1 off



RESET BUTTON

On the back of the POWER CARTRIDGE there is a Reset Button. Pressing this button makes a SPECIAL MENU appear on the screen.

This function will work with many programmes.

CONTINUE	Allows you to restart the interrupted program
BASIC	Returns to BASIC
RESET	Simulates RESET
TOTAL	Erases the contents of the memory, like a Disk. The program can be a reset later with RESTART followed by CONTINUE
BACKUP	RESET at any program
DISK	As BACKUP DISK but no TAPE
TAPE	
HARDCOPY	At any moment prints out a Hardcopy of the screen. Entering CONTINUE afterwards you can return to the program
MONITOR	Takes you into the Machine language Monitor

BDL

Bitcon Devices Ltd

88 BEWICK ROAD
GATESHEAD
TYNE AND WEAR
NE8 1RS
ENGLAND

Tel: 091 490 1975 and 490 1919 Fax 091 490 1918
To order: Access/Visa welcome · Cheques or P/O payable to BDL
Price: £16.99 Incl. VAT.
UK orders add £1.20 post/pack total - £18.19 Incl. VAT.
Europe orders add £2.50 Overseas add £3.50
Scandinavian Mail Orders and Trade enquiries to: Bhab Elektronik, Box 216, Norrtalje 76123, SWEDEN. Tel: +46 176 18420 Fax. 176 18401
TRADE AND EXPORT ENQUIRIES WELCOME

CONTENTS

Commodore DISK USER
Volume 4 Number 4 FEBRUARY 1991

ON THE DISK

IN THE MAGAZINE

Publisher: Hasnam Walji
Group Editor: Paul Eves
Technical Editor: Jason Finch
Publishing Consultant: Paul Crowder
Advertisement Manager: Cass Gilroy
Designer: Mark Newton
Distribution: Seymour Press Distribution Ltd, Winsor House, 1270 London Road, Norbury, London SW16 4DH. Tel:081 679 1899. Fax: 081 679 8907
Printed By: Gibbons Barford Print

Subscription Rates

UK	£33.00
Europe	£39.00
Middle East	£39.30
Far East	£41.60
Rest of World	£39.70 or $69.00

Airmail rates on request
Contact: Select Subscriptions. Tel: (0442) 876661

Commodore Disk User is a monthly magazine published on the 3rd Friday of every month. Alphavite Publications Limited, 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Telephone: (0908) 569819 FAX (0908) 260229. For advertising ring (0908) 569819

EDITORS COMMENT

As I expected, I have had one or two complaints concerning the DECEMBER 1990 issue of CDU. In all cases the problem was the same, why so many programs for the C128 and only 2 for the C64. What a lot of you seem to forget is that CDU is NOT just devoted to the C64. True, in the main most of the content and programs are C64 orientated (This is because we obviously get more submissions from C64 users), however, what you MUST remember is that CDU is for both C128 and C64 users. I have a responsibility to ALL readers/users be they C128 or C64. Therefore, from time to time I have to cater for all the C128 people (who normally don't get a look in). Likewise, that responsibility is also concerned with GAMES players as well as SERIOUS users. So, once again, every time I include a GAME on the disk I get phone calls and letters of complaint. Just like the old saying goes;

YOU CAN PLEASE SOME OF THE PEOPLE ALL OF THE TIME
YOU CAN PLEASE ALL OF THE PEOPLE SOME OF THE TIME
BUT YOU CANNOT PLEASE ALL OF THE PEOPLE ALL OF THE TIME

Just because you get ONE issue which doesn't happen to appeal to you personally, doesn't mean you have to cancel your subscription. At least, I don't think it does. Anyway folks! I have said my peace, I will leave it up to your own judgement as to whether you think CDU is value for money or not. I personally think it is. (So do some 30,000 other people)

DISK INSTRUCTIONS

Although we do everything possible to ensure that CDU is compatible with all C64 and C128 computers, one point we must make clear is this. The use of 'Fast Loaders', 'Cartridges' or alternative operating systems such as 'Dolphin DOS', may not guarantee that your disk will function properly. If you experience problems and you have one of the above, then we suggest you disable them and use the computer under normal, standard conditions. Getting the programs up and running should not present you with any difficulties, simply put your disk in the drive and enter the command.

LOAD" MENU",8,1

Once the disk menu has loaded you will be able to start any of the programs simply be selecting the desired one from the list. It is possible for some programs to alter the computers memory so that you will not be able to LOAD programs from the menu correctly until you reset the machine. We therefore suggest that you turn your computer off and then on again, before loading each program.

HOW TO COPY CDU FILES

You are welcome to make as many of your own copies of CDU programs as you want, as long as you do not pass them on to other people, or worse, sell them for profit. For people who want to make legitimate copies, we have provided a very simple machine code file copier. To use it, simply select the item FILE COPIER from the main menu. Instructions are presented on screen.

DISK FAILURE

If for any reason the disk with your copy of CDU will not work on your system then please carefully re-read the operating instructions in the magazine. If you still experience problems then:

1. If you are a subscriber, return it to:
Select Subscriptions Ltd
5, River Park Estate
Berkhamsted
Herts
HP4 1HL
Telephone; 0442 876661

2. If you bought it from a newsagents, then return it to:
CDU Replacements
Interceptor Group
Mercury House
Calleva Park
Aldermaston
Berks
RG7 4QW
Telephone; 0734 817421

Within eight weeks of publication date disks are replaced free.

After eight weeks a replacement disk can be supplied from INTERCEPTOR GROUP for a service charge of £1.00. Return the faulty disk with a cheque or postal order made out to INTERCEPTOR GROUP and clearly state the issue of CDU that you require. No documentation will be supplied. Please use appropriate packaging, cardboard stiffener at least, when returning disk. Do not send back your magazine, only the disk please.

NOTE: Do not send your disks back to the above address if its a program that does not appear to work. Only if the DISK is faulty. Program faults should be sent to: BUG FINDERS, CDU, Alphavite Publications Ltd, Unit 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Thank you.

THE CRANMORE DIAMOND AFFAIR

Steal the diamond in 9 hours TONY ROME

Your hotel room was ideally suited to the audacious scheme that you had dreamed up. Across the street stood CRANMORE building where the famous CRANMORE DIAMOND was being exhibited.
You reflected how your present circumstances had forced you to consider stealing the precious gem, but that was the reason you were here!
You were in a small town with a few shops, a hotel and a police station. You certainly had no intention of ending up there. You glanced at the time, it was almost 5.00 pm, and the last day of the exhibition. You had about 9 hours to execute your daring plan. After that, the diamond would no longer be accessible.
You stared vaguely out of the hotel window, below street lamps everywhere glowed brightly in the evening dusk. One sip of Brandy you thought to calm your jangling nerves, and then it was time to begin.....

The **CRANMORE DIAMOND** affair is a text and graphic adventure that tests all your powers of nerve and deduction. Throughout the adventure you give your commands in the usual way. For example, Take the Paper or Examine the Chest. Etc Etc. The adventure is set in real time., providing you have a WATCH Some of the commands you already know are,

TAKE/GET - To acquire an object
DROP - To discard an object
N/S/E/W - To move around the adventure
LOOK - To view your current location
TEXT/WORDS - To follow the adventure in text only
PICT - To follow the adventure with graphics also
LIST/INV - To display you belongings
SAVE - To save current position in the game
LOAD - To reload a previously saved position
TIME - To restart the real time display throughout the game
'X' - To cancel the real time display

You may also be able to converse with certain people For example; 'Ask waiter about the drink' Etc. A feature of the game is the ability to make fairly complex commands like; Take the watch and throw it. A word of warning. Look out for the Thief!! Good luck on your quest.....

POWER COMPACTOR

A useful routine for those long programs MARTIN PIPER

One of the biggest problems many programmers face these days, and days gone by, is that their masterpieces are just too big for the computer's memory. As programs get more and more sophisticated, users are demanding more and more options from them. This in turn provides the programmer with the Catch-22 situation of wanting to provide, but not enough memory space. This is were POWER COMPACTOR comes to the rescue.

WHAT IT'S ALL ABOUT

This is a nice, short and sweet program that enables you to pack files into one file that will, upon loading, decompact it and RUN it. Once loaded you are presented with the file entry screen. On this screen you are asked to insert the disk with the files you want to pack, then you press RETURN. The drive will then load in the directory. You then SCAN the files you want with the CURSOR UP/DOWN keys and press RETURN on the files you require. On your last choice press RETURN on the white QUIT bar to edit your choices (in case you made a mistake in entering the files).

After that you enter the start address to RUN the program. This is the SYS call for the machine code to run it. The drive will then whirr away packing all the files into one, after this is finished it will ask you for a file name to SAVE and ask for the correct disk you wish to SAVE to. After the program is saved the packer will RESET. The usual overhead is two disk blocks but this is made up as the file will take a shorter time to load in.

NOT FOOLPROOF

Please note that this program is not fool proof, as if the end of one file overwrites the start of another file, the packed file will get corrupted. For example; If a piece of GRAPHICS loads in at $0800 and lasts for 9 disk blocks (that is, it ends at $1100) and a MUSIC file is packed and that one starts at $1000, then problems are found as the GRAPHICS will overwrite the MUSIC. There is also a maximum size that can be compacted. This is 30K which should be big enough for most needs. Also note, the maximum size of the DIRECTORY should be 2K

THE 64's KEYBOARD

Look after your keyboard BONES

We present an in depth look at the keyboard of the C64 and explore some of it's possibilities, including care and maintenance

The Commodore 64 has been equipped with a very reliable keyboard which comprises of 66 keys. The keys can be removed quite easily simply by inserting a flat object, such as a small screwdriver, beneath the key and gently easing it off. Once off you will discover a small coiled spring that is located underneath which helps to remove 'clack' when you are typing. The spacebar is somewhat less easy to remove. It incorporates a metal stabilizing rod which locates within two lugs, and should you find the need to remove it, approach it with some care

Faulty or damaged keys can be replaced quite easily, and the top of the computer housing cleaned Whilst cleaning the keypads themselves be careful when using certain types of cleaning agents, they can easily remove the white paint of the graphic symbols - I speak from annoying experience.

INTERRUPTS AND THE KEYBOARD

During an interrupt phase (which occurs every 50th of a second) the keyboard is scanned and any keypress which may be occurring is updated to the keyboard buffer. The keyboard buffer is an area of memory allocated to hold the ASCII code of up to ten consecutive key presses.

When an interrupt occurs the 6510 processor finishes dealing with it's current instruction, and then saves several important values after which it jumps to the top of memory at address $FFFE to test if the interrupt has been generated by the BRK instruction If it is not, and therefore a normal interrupt It will then jump to the address which is stored at memory locations $314/315 (788/789). This is normally set to $EA31 in the usual form, High/Low byte As it is situated within RAM this address can be changed to point to user routines - such as 'split-screens', 'multiple sprites', 'special effect and sounds' or whatever. The user routine will usually end with a jump to $EA31 and the processor will continue with it's normal interrupt sequence.

The system clock is updated, the run/stop status is saved, then screen and tape handling (i.e. Cursor flash - switch off cassette motor unless a flag is set to say otherwise), finally it calls the SKNKEY routine at $FF9F. This address is within the kernal jump table, which redirects the program to the ROM routine starting at $EA87

SCANNING THE KEYS

Only 64 of the 66 keys are actually scanned. The keyboard is hard-wired to form an 8x8 matrix, or grid. When a key is depressed this will short a coordinate upon the grid giving it a value of zero. Two bytes are used to hold the coordinate values. Eight columns are held at memory location $DC00 (56320), and eight rows use $DC01 (56321). (You may recognize these as the two ports for Joystick/Paddles.) Hence a total of 8x8 = 64 locations for the 64 keys. The character positions within the 8x8 matrix can be examined if you refer to TABLE 1. The two keys which are not read in this manner are the RESTORE key and the SHIFT/LOCK key, these are handled separately

PROCESSING THE SCAN

The columns $DC00 (56320) are set for output and the rows $DC01 (56321) are set for input A loop of eight iterations is initiated, and within each iteration $DC01 (56321) (rows) is rotated through each of it's eight 'bits'. If the bit is set (= 1) then a key in that row has been depressed, otherwise the bit is reset (= 0). The routine will now check to see if the key is either SHIFT, C=, or CTRL. If it is then the register which stores this information (SHFLAG $28D (653)) is updated. (Note: These can be detected and usefully exploited by using PEEK(653) 1 = SHIFT key, 2 = C= key (COMMODORE), 3 = CTRL key. These values add together if more than one key is simultaneously depressed, e.g SHIFT/CTRL = 5, C=/SHIFT = 3, etc.).

If any other key is pressed the value of the counter (0 at start and incremented upon each test), is stored into Register SEDX $CB (203).

After each row is processed the next column iteration is performed until all 64 keys have been examined

KEY TABLES

When the keypresses are converted into ASCII, the 64 has four tables set up in ROM. These are: (A) Unshifted characters, (B) Shifted characters, (C) C= (Commodore)/keys, and (8) CTRL/keys. The tables are located, respectively, from $E881, $EBC2, $EC03, and $EC78. Each table is 64 bytes long and uses the value of the counter, stored in register LSTX $C5 (197) to point to the appropriate ASCII value contained within the current table in use. The final, or 64th location in each table is set to $FF (255) to denote that no key has been pressed

(remember, the counter starts a zero, so 64 iteralions will take the counter to 63, if no key has been pressed then the final value of the counter will be 64)
PROGRAM 2 illustrates the way in which the Columns and the Rows interact with each other At this stage there is no distinction between shifted/C=/unshifted/CTRL keys.

CLOSING STAGES OF THE INTERRUPT

A further routine is entererl at $EAE0 where the ASCII value of the key is evaluated, with where repeats and cursor control are taken care of. At this point the program jumps to a new locatton to conclude the keyscan, however, it uses a vector, KEYLOG $28F/$290 (655/656), trom which to oblain the address to jump to. It is at this point where we are able to 'wedge' in a user routine lo intercepl lhe keys to implement messages or whatever. See PROGRAM 3 for an example of this method.
Finally, LSTX $C5 (197) copies the value In SFDX $C8 (203), LSTSHF $28E (654) copies SHFLAG $28D (653), and the keyboard buffer is updated. $DC00 is set to it's default value of $7F (127), and the routine terminates with the recovery of the values which it saverl at the start, then a return from the interrupt, and the continuation of whatever it was doing before the interrupt occurred.

LOCATING KEYS

A useful method for finding a keypress as opposed to the more usual Basic GET statement is to access the bytes which store lhe counter number or the most recent keypress. This is stored at memory locations $C5 (197), AND $CB (203).
During 'keyscan' routine, and if a key has been struck, then an identifying number is placed into location $CB (203]. The previous keypress is stored in $C5 (197). Comparing these two locations will show if a new key has heen struck
Using the method of accessing etther $C5 (197) or $CB (203) does not take into account shifted characters, only the physical key. The default value is 64 which is representative of no key being struck.
If you only require the physical key and are not concerned with shifted keys then memory location $C5 (197) can be used. PROGRAM 1 illustrates a method of accessing $C5 (197). Remember lo use the values outline in TABLE 2 and not ASCII. If you do need to distinguish between shifted/unshifted keys then check $D7 (215) for the ASCII value of the last key pressed. Once again SHIFT/C=/CTRL can be detected separately from the content of $28D (653).

DISABLING RUN/STOP AND RUN/STOP RESTORE

METHOD A
1) POKE 808,54: POKE 809,188 - Disables RUN/STOP and RUN/STOP RESTORE
2) POKE 808,237: POKE 809,246 - Re-enables
(This leaves the system clock working, does not affect tape operations, and LIST will work quite normally)

METHOD B
1) POKE 808,234 - Disables RUN/STOP and RUN/STOP RESTORE
2) POKE 808,237 - Re-enables
(This method will scramble the LIST and may have atfects on tape loading. Okay if you are not doing any tape operations)

METHOD C
1) POKE 788,52 - Disables RUN/STOP
2) POKE 788,49 - Re-enables
(This is okay - it doesn't disable RESTORE but will, during lape operations, reset everything bark to normal)

METHOD D
1) POKE 792,193 - Disables RUN/STOP RESTORE only
2) POKE 792,71 - Re-enables

REPEAT KEYS

Location RPTFLG $28A (650) is the byte which controls which keys will repeat if a key is held down. To modify this function use:

POKE 650,65 - Turns off all key repeats
POKE 650,128 - Turns on all key repeats
POKE 650,0 - Turns on Spacebar and Cursor controls

PROGRAM 1
This small program uses memory location $C5 (197) to test if etther function key F1 or F3 has been struck lo execute colour changes to screen/border

```
10 KP=PEEK(197)     ;SET A VARIABLE, KP, EQUAL TO
15 :                ;THE CURRENT VALUE HELD IN
16 :                ;LOCATION 197 ($C5)
20 IE KP=64 THEN 10    ;NO KEY HAS BEEN PRESSED
25 :                   ;LOOP UNTIL PRESSED
30 IE KP=4 THEN BC=BC+1 ;KEY F1 = 4 (SEE TABLE 2)
40 IE BC=16 THEN BC=0  ;KEEPS 8C BETWEEN 0 AND
15
50 IE KP=5 THEN SC=SC+1 ;KEY E3 = 5 (SEE TABLE 2)
60 IF SC=16 THEN SC=0  ;KEEPS SC BETWEEN 0 AND
15
70 POKE 53280,BC       ;UPDATE BORDER COLOUR
80 POKE 53281,SC       ;UPDATE SCREEN COLOUR
90 GOTO 10             ;LOOP 8ACK TO DO SOME
                        MORE
```

PROGRAM 2
This program demonstrates the way in which the columns ($DC00) and the rows ($DC01) interact with each other during the Keyscan routine.

```
10 FOR X=O TO 17· READ D  ;POKE ML ROUTINE
TO SPARE
15 POKE 49152+X,D:NEXT  :RAM FROM 49152
($C000)
20 POKE 788,52  ;DISABLF STOP KEY
30 INPUT "($DC01) COLUMN NO:";C ;TEST A
COLUMN (TABLE 1)
40 POKE 49153,C  ;POKES IN COLUMN  VALUE
50 SYS 49152   ;CALLS ML ROUTINE
60 GOTO 50  ;RUN/STOP RESTORF TO  STOP
70 DATA 169,0,141,0,220,174,1,220
80 DATA 169,0,32,205,189,169,13,76
90 DATA 210,255
```

The Basic data statements of lines 70-90 in Assembler are:

```
C000  LDA #0 ;THE BASIC INPUT WILL FILL $C001
WITH COLUMN
C002  STA $DC00 ;VALUE 127,191 ETC AND STORE IT
AT $DC00
C005  LDX $DC01 ;GET THE ROW VALUF FROM
$DC01 - 255=NO KEY
C00B  LDA #0 ;FILL ACCUMULATOR WITH NULL BYTE
AND JUMP TO
C00A  JSR $BDCD ;THE ROM ROUTINE WHICH WILL
PRINT VALUE OF X
C00D  LDA #13   ;PUT VALUE FOR CARRIAGE RETURN
TO ACCUMULATOR
C00F  JMP $FFD2 ;AND FXECUTE RETURN FROM
KERNAL ROUTINE CHROUT
```

PROGRAM 3

Basic Keyword Printer... This example program brings together a method to output to the screen full basic keywords, using only single key inputs. See TABLE 3 for a list of the keywords and their Key assignments.

```
10      *=$C000
12      ; EQUATES
20      VEC=$2BF ;$2BF/290 ARE THE HI/LO BYTES
        FROM WHERE
25      ;THE FINAL PART OF 'SKNKY' ROUTINE
        GETS ITS START ADDRESS
30      KEYSCAN=$EB48  ;THE FINAL PART OF ·
        'SKNKY' START ADDRESS
40      BASICWRD=$AO9C ;START ADDRESS OF
        BASIC KEYWORD TABLES
50      CHROUT=$FFD2   ;KERNAL ROM ROUTINE
        TO OUTPUT A CHARACTER
5S       ;IN THIS CASE, PRINT CHAR ONTO THE
        SCREEN
60      SFDX=$CB;PREVIOUS KEY PRESS
70      LSTX=$CS ;LATEST KEY PRESS
B0      SHFLAG=$2BD;BYTF TO TEST FOR
        SHIFT/C=/CTRL KEYS
120     SETUP
130     LDA #<GETWORD ;PUT THE HI/LO BYTFS OF
        GETWORD IN
140     STA VEC ;TO VECTOR TO REDIRECT
        SKNKY TO
1S0     LDA #>GETWORD ;OUR WEDGE. SETUP
        WOULD BE CALLED
160     STA VEC+1;FROM BASIC WITH <SYS 49152>
170     RTS
190     GETWORD
200     LDA SHFLAG ;TEST THE LAST CONEIG OF
        SHFLAG TO
210     CMP #4;DETECT IF THE CTRL KEY IS PRESSED
220     BNE EXIT   ;IF NOT THEN EXIT BACK TO
        SKNKEY
230     LDY SFDX   ;ELSE LOAD THE Y INDEX WITH
SFDX TO
240     CPY #64    ;TEST FOR KEYPRESS.
250     BEQ EXIT   ;NO KEYPRESS SO EXIT TO SKNKY
2B0 ;IF A KEY HAS BEEN PRESSED, TEST IT AGAINST
THE PREVIOUS
290 ;KEYPRESS AND IF IT IS THE SAME KEY THEN EXIT
SKNKY. THIS
300 ;STOPS THE KEYWORD FROM REPETITION IF THE
KEY IS HELD
310 ;OVERLY LONG.  HOWFVER, IF IT IS A NEW KEY
THEN STORE IT
320 ;IN LSTX. SET THE X INDEX TO ZERO READY TO
TEST EACH KEY
330 ;WORD TO FIND THE LAST CHARACTER. THE Y
INDEX WILL BE
340 ;USED TO COUNT OFF EACH KEYWORD FROM
THE TABLF
360         CPY LSTX
370         BEQ EXIT
3B0         STY LSTX
390         INY
400         LDX #0
430 ;AFTER THE ASCII CODE FOR THE CHARACTER
HAS BFEN
440 ;TRANSFERRED FROM MEMORY TO THF A REG
THE ROUTINE TESTS
450 ;THE BTH BIT OF THE BYTF. IF THIS IS SET (=1)
THEN THF
460 ;END OF THE KEYWORD HAS BEEN REACHED.
DECREMENT Y INDEX.
470 ;WHEN Y REACHES ZERO THEN THE X INDEX
POINTS TO THE LAST
4B0 ;CHARACTFR OF THE BASIC WORD SELECTED.
490 ;
500 LOOP1
510         INX
520         LDA BASICWRD,X
530         BPL LOOP1
540         DEY
550         BNE LOOP1
560         INX
590 ;THIS SECTION OF THE CODF WILL TEST THE
BTH BIT OF EACH
600 ;BYTE TO FIND THE LAST CHARACTER OF THE
BASIC WORD. IF IT
610 ;IS THEN THE BIT WILL BE SET SO BRANCH TO
LASTCHAR
630 ;ELSE PRINT CHAR TO SCREEN USING ROM
ROUTINE CHROUT, THEN
640 ;LOOP TO GFT NEXT CHARACTER BYTE.
660 LOOP2
670         LDA  BASICWRD,X
6B0         BMI  LASTCHAR
690         JSR CHROUT
700         INX
710         BNE LOOP2 ;X WILL NEVER REACH ZERO.
```

```
720 ;SO ALWAYS BRANCHES.
750 ;ONCE WE HAVE THE LAST CHARACTER WE
MUST AND IT WITH 127
760 ;TO TURN OFF THE 8TH BIT THEN OUTPUT THE
CHARACTER TO THE
770 ;SCREEN AND EXIT TO SKNKY ROUTINE.
790 LASTCHAR
800      AND #$7F
810      JSR CHROUT
820      EXIT
830      JMP KEYSCAN
850      END
```

Here is the same program as a Basic Loader for those who do not have an assembler.

```
100 DATA 169,11,141,143,2,169,192,141
110 DATA 144,2,96,173,141,2,201,2
120 DATA 208,41,164,203,192,64,240,35
130 DATA 196,197,240,31,132,197,200,162
140 DATA 0,232,189,156,160,16,250,136
150 DATA 208,247,232,189,156,160,48,6
160 DATA 32,210,255,232,208,245,41,127
170 DATA 32,210,255,76,72,235
180 EOR X=0TO61:READ D: POKE 49152+X,D: NEXT
```

To activate KEYPRINT use SYS 49152
To exit KEYPRINT hold down RUN/STOP and strike RESTORE

(TABLES 1,2 and 3 are bottom right and over the page).

TABLE 4 - SCREEN LOCATIONS

STKEY $91 (145) RUN/STOP key
LSTX $C5 (197) Latest Key press
NDX $C6 (198) Number of characters in keyboard buffer
RVS $C7 (199) Reverse flag
SFDX $CB (203) Previous key pressed
BLNSW $CC (204) Cursor blink enabled
BLNCT $CD (205) Timer countdown to toggle cursor (12-0)
GDBLN $CE (206) Character under cursor
BLNON $CF (207) Cursor blink flag (0or1)
CRSW $D0 (208) Flag:input from screen or keyboard (3or0)
PNTR $D3 (211) Cursor column on current line
TBLX $D6 (214) Cursor row number
$D7 (215) ASCII value of key pressed
COLOR 286 (646) Current character colour code (0to15)
HIBASE $288 (648) Top of scrn page memory (usually 4)
XMAX $289 (649) Max No of chars in keyboard buffer
RPTFLAG $28A (650) Repeat Flag (0 = space/cursor: 64 = no keys: 128 = all keys
KOUNT $28B (651) Repeat delay (4-0: 12 repeats per second)
DELAY $28C (652) Repeat countdown :16-0 secs before repeat
SHFLAG $28D (653) Shift/Commodore key/CTRL register (1,2,4)
LSTSHF $28E (654) Previous configuration of SHFLAG
MODE $291 (657) Commodore key/Shift mode switch on/off (128 = off)

VECTOR POINTERS

KEYTAB $F5/$F6 (245/246) Keyboard table pointer
LXSP $C9/$CA (201/202) Cursor row/column position at start of input
PNT $D1/$D2 (209/210) Current screen line address pntr
USER $F3/$F4 (243/244) Pntr to current colour ram address
KEYLOG $28F/$290 (655/656) Vector for implementing 'wedge'

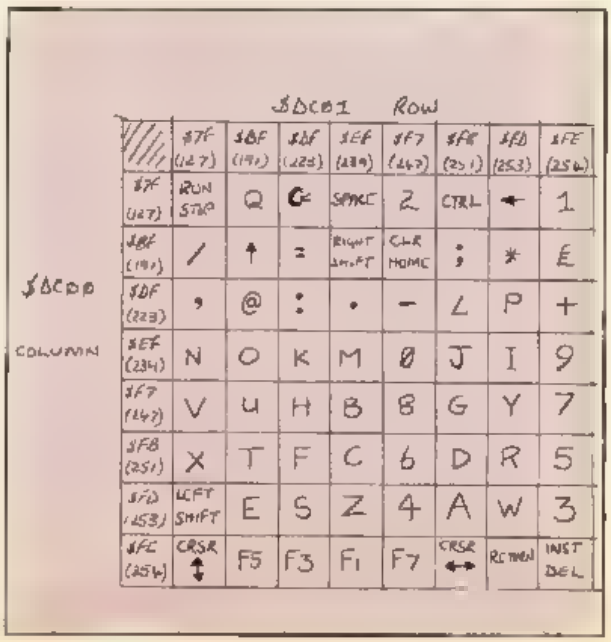
KEYBOARD BUFFFR

KEYD $277-$280 (631-640) Keyboard buffer

ROM ROUTINES

$E544 (58692) Clear screen
$E5A8 (58792) Set VIC chip to defaults
$E632 (58930) Input arrives here
$E8EA (59626) Scroll up one character row
$E981 (59777) Scroll down one character row
$E9FF (59903) Clear entire row (e.g. POKE 781,n:SYS 59903 Ô2 where n is set equal to row number (0-24)
$EA31 (59953) Interrupt sequence starts here
$FFD2 (65490) Output a character to device (default screen)
$FFE4 (65508) Get character from the keyboard queue (buffer)

TABLE 1 - KEYBOARD DECODING GRID



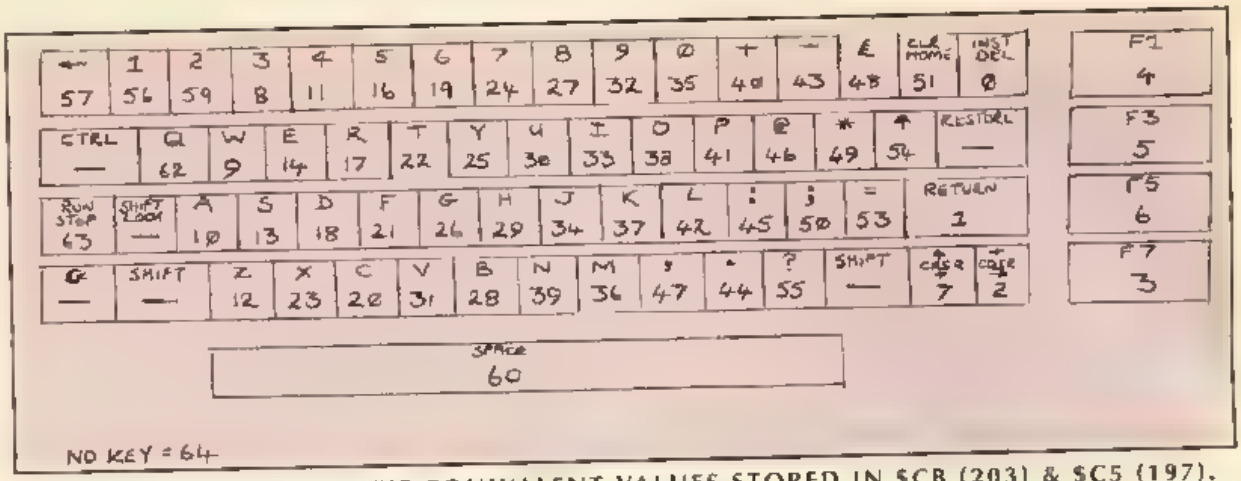


TABLE 2 - KEYS AND THEIR EQUIVALENT VALUES STORED IN $CB (203) & $C5 (197).

← POS	CTRL —	RUN/STOP SIN	Z RESTORE	F1 INPUT#
1 FRE	Q COS	SHIFT/LOCK —	X POKE	F3 INPUT
2 RND	W GOTO	A RUN	C SAVE	F5 DIM
3 LET	E RETURN	S GOSUB	V OPEN	F7 DATA
4 IF	R ON	D WAIT	B CLR	
5 STOP	T DEF	F VERIFY	N THEN	SPACE-BAR LOG
6 LOAD	Y PRINT	G CONT	M TO	
7 PRINT#	U SYS	H CMD	, AND	
8 LIST	I GET	J NEW	. *	
9 CLOSE	O SPC(	K FN	/ USR	
Ø TAB(	P STEP	L +	CRSR ↕ READ	
+ NOT	@ ↑	: /	CRSR ↔ NEXT	
- —	* >	; =		
£ OR	↑ ABS	= INT		
CLR/HOME <	RESTORE —	RETURN FOR		
INS/DEL END				

TABLE 3 - KEYS TO BASIC KEYWORDS

MULTITASKING C128

EXPANSIONS END THE SERIES

DAVID KELSEY

Last month we gave you the file necessary to display all the ASSEMBLER files that we have been discusing. This month sees the series coming to and end with the emphasis on EXPANSIONS to the system

PRG TO PRG COMMUNICATIONS

This facility will be very useful. It allows data to be transferred between programs. The way to transfer information would be via operating system routines which you could then give information and a program name. This information would then be stored in a table. Another routine could then be called it could interrogate the operating system to see who called him, then pass any messages back to the caller. A use for this would be a DISK control system. Another program could then send information to disk or receive information from disk via the application program rather than to provide a full operating system extension. The operating system providing control purely over the CIA chips.

The way I envisage this being applied is by using a fixed area within zero page to either place messages or receive data When the routines are called, because they will use the applications zero page and page 1, there isn't a problem. These roulines will have access to a large block of memory where the data and the application name to whom the information is destined. This table could be chained and whenever a new message is created, a bit of free storage can be got to add the new data block There is a danger with memory being used when a program sends data to another and either that program isn't receiving the data because it isn't working or the application is not loaded

PRIORITY

For some reason or another you may want a program to get more CPU time than another. This means the program will execute quicker than other programs. A simple facility to load programs with a priority is already provided, but the facility isn't used. An algorithm is required within the routine that selected the next program to run. It would select the most likely program to run based on the priorities of all the programs rather than just select next one. It isn't a viable solution to just let programs with higher priorities run longer because this is noticeable. An experiment that I did had 3 screen locations maintained by different programs being updated. Letting one program run for more time that the others gave the picture of one program running for 3 seconds the next 2 ran for 1/2 a second each. This isn't what is required. The aim is to have a program look as though it is running fast while others around it run slowly. In the above example what should be seen is one screen location should be changing rapidly while the others also change but at a slower rate (a similar situation could be achieved by writing 3 programs that update the screen, but put delays into 2 of them).

INTERRUPTS

An application cannot use interrupts as the operating system uses the system interrupts. It would be dangerous to trap the system interrupts before passing control back to the operating system routines At present the operating system thinks that any interrupt is intentional and meant for him. What would be nice is if an application can set up his own vectors for IRQ and NMI. This could be achieved by calling an operating system routine to define vector addresses (obviously these addresses will either have to be given when the program is run and the program itself will have to work out where his is really or at load time when the addresses used, work out at assembly time, will have to be relocated). The only problem will be when an interrupt occurs, who gets the interrupt. The operating system can simulate the execution of a vector by changing where the next executable instruction is found by altering the PC on the stack when the interrupt is returned. Once the interrupt code has been completed, how do we get the operating system to recognise this and so return to the correct point in the application? Maybe an extension to BRK processing could do this (see later). This still leaves the

problem of who gets the interrupt. As there are several different interrupts, maybe different interrupts can be dedicated to different applications (ie if a certain type of interrupt occurs, then only one program will ever be informed). There may be a way in queueing programs to get the same interrupt. Eg 2 programs want the same interrupt, either they both receive it or the first program gets the first interrupt and the second program gets the second interrupt

80 COLUMN SUPPORT

As this is a screen, there are problems with several applications using it. It may be necessary to dedicate this device purely to one application. This could be done by allowing the first caller to the routines to only use the screen. If he ever stops running, it could free up the screen for another application to use.

D O S

This would manage all transfer of data to and from the disk drive. It would allow several applications to be able to have different files being accessed at the same time on the one drive. Eg, one program writing a file while another program accesses another file to be outputted to the printer. This could run either as an application or be done within the operating system using operating system routines. You would have to use IPC (Inter Program Communications) to talk to the application version.

RESTORE KEY

At present, If you press restore to get control of the operating system, the whole process stops. It would be nice to communicate with the operating system without all other applications stopping. The reason it stops at the moment is that if the command processor was interrupted, and another program was run, because the command processor routines are not in the program list, the command processor is never returned to when the system locates the next program to run. Obviously you need some way of making sure that the routine returns to the command processor while running other programs

BANK LOCATION

The operating system code by virtue of the way it was designed must be below $4000 as the interrupt routines set up by the system, set the memory configuration to $00 which means all BASIC ROMs are enabled. It would be nice to have a system that didn't rely on this and could be placed anywhere in RAM, even RAM block 1.

TRACING AND DIAGNOSTICS

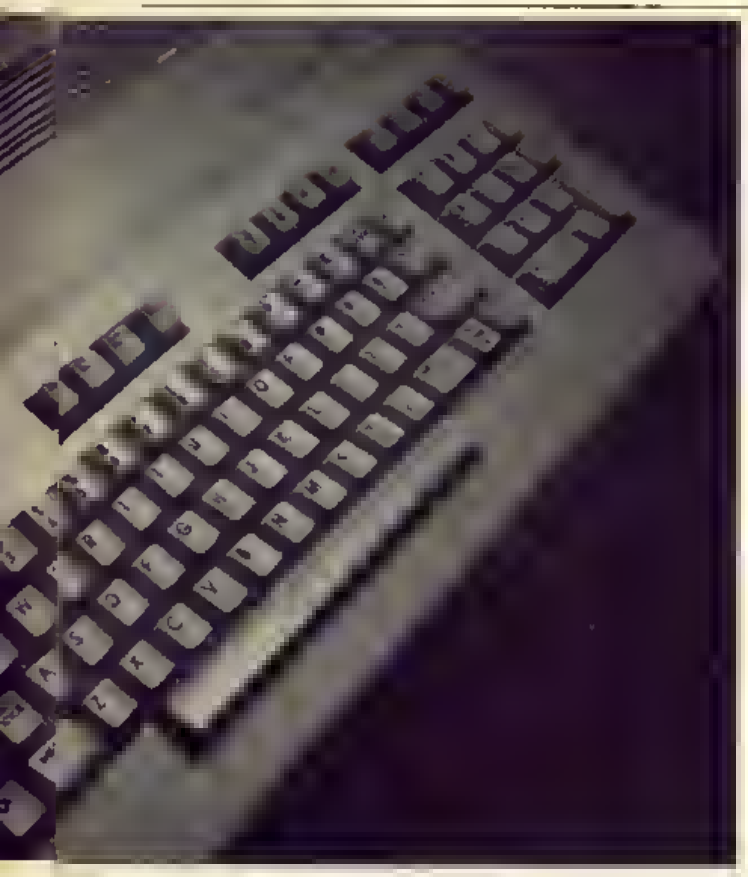
If an application doesn't execute properly, it would be handy to be able to trace a program to detect errors. Also if an application causes the operating system to crash it would be useful to have in memory a table listing exactly what the operating system was doing before it fell over. A program could be loaded (without the operating system) to scan memory for this table and display the information found there. These ideas would provide diagnostics for both applications and the operating system. (information such as the registers should also be in the table).

EXTENSIONS TO BRK COMMAND

At present, BRK just terminates a program however it could be used to perform different functions depending in the value of the Accumulator. An example has already been given above. Another could be setting A=0 and issuing BRK to terminate a program. Other ideas could be to provide special functions controlled by the operating system rather than going through system routine calls.

IMPROVING THE RELOCATION ROUTINE

The relocation routine is very basic and limits the programmer to the design of programs. It could be possible to provide extra facilities within the relocation code so as to allow programming features such as address tables. To do this may require a special programming technique which is defined by exta code in the relocation routine.



Z80 PROGRAMS

It may be possible to switch processors at interrupt time also, thus allowing programs coded in Z80 to be used as well. But this would require quite a bit of research.

ADDING TASKS, LOADING OTHER PROGRAMS

You may have heard the term SUBTASK. This is where a program running under a multitasking operating system, has within it a section of code that runs independently of the main body of the program. It is like 2 programs within one that run together. At present I have no way of implementing this. Another useful feature would be an application which can load other applications into the system.

POST AND WAIT

Another concept within multitasking environments is POST and WAIT. I shall explain this by example. There are several programs running, and one issues a WAIT for another program. The program then doesn't do anything until the program he is waiting for, issues a POST for that program. Once the POST has been detected, the original program starts processing again. The feature is useful if you have one program that must wait for another to complete a function before he can process the correct information.

REMOVING DEPENDENCE ON OLD OPERATING SYSTEM

The operating system still uses the original operating system's LOAD and KEYBOARD routines. This should not be the case and needs to be removed by coding own serial and keyboard routines which could be used by application programs.

OTHER IDEAS

You could use CIA timers to control when an IRQ interrupt occurs. Thus varying the length of time a program runs for.

APPLICATIONS

These are programs that could run under the multitasking operating system.

ASSEMBLER

A macro assembler could be designed to allow for coding restrictions imposed earlier by the operating system. It could also provide easy ways to access the operating system routines designed for use by applications.

EDITOR

An editor will be required to build files. These files could be processed by the assembler. The beauty here is that while the assembler is assembling one program, you could be editing another. This is the advantage of multitasking operating systems.

PRINTER CONTROL

I have always hated having to wait for the printer to finish before I can do any further work on the computer. What would be nice is to tell the print program to print information off the disk while you can use another application.

CONCLUSION

I have found this a very interesting project and it has taught me a lot about multitasking operating systems. I hope you have found this article interesting as well. There is much development that can be done and more to discover about multitasking. I hope this article will start some further development in this field.

ADVENTURE HELPLINE

More clues for The Astrodus Affair JASON FINCH

Welcome to the second part of the ASTRODUS Adventure Helpline which will be running for another few months, hopefully informing you of how the excellent adventure, the ASTRODUS Affair, can be successfully completed without you needing to hurl large objects at your monitor and computer. Last month I promised to give you the rest of the location descriptions and a taster of the vocabulary. So below is exactly that, starting with the location that we left off on last month, number fifteen.

15

You're in a plain, compact corridor leading directly west to east. A small silver disc is set into the floor.
EXITS: EAST 16, WEST 14

16

This is cross-section A, with exits in all four cardinal directions.
EXITS: NORTH 12, EAST 17, (SOUTH 19), WEST 15

17

You're at the base of a steep metal stairwell leading into the darkness above. Another exit leads west.
EXITS: WEST 16, (UP 2)

18

Banks of lights and displays illuminate the walls of this surprisingly dark control room, with the main console totally filling one wall. A door leads north.
EXIT: NORTH 14

19

You're at the top of a short flight of steps leading downwards. An exit leads to the north.
EXITS: NORTH 16, DOWN 20

20

You're in cross-section B. Corridors branch off to the west, east and south, and a flight of stairs leads upward
EXITS: EAST 24, SOUTH 22, WEST 21, UP 19

21

This is the main supplies room, although anything of value has already been pillaged. A single door leads east.
EXIT: EAST 20

22

You find yourself in cargo bay 2. Damaged crates and cargoes are strewn about the floor. Doors lead north and south. A transporter is situated in one corner, but it is damaged beyond repair.
EXITS: NORTH 20, (SOUTH 23)

23

This is cargo bay 1. Virtually empty compared to bay 2, this contains a few broken crates, and a rather hefty safe. A door leads north.
EXIT: NORTH 22

24

You're in a long grey corridor running east to west. At the eastern end there is a door, with an alpha-numeric keyboard beside it, and a visual display above it. The door is open/closed.
EXITS: (EAST 6), WEST 20

VOCABULARY

Now that all the location information is out of the way we can move on to the vocabulary. It would be pointless my simply listing every noun and every verb that the program recognises - instead I shall cover just a few of each. Some of the more unusual verbs are as follows: ACTIVATE, CRAWL, DRAG, ELECTROCUTE, ENTICE, INPUT, RECHARGE, REPLACE, ROTATE, SWING AND SWITCH.

You should find that that small list will help in a few of the specialised situations, as will the following verbs: ANTENNA, DRYGARS, GENERATOR, KEYBOARD, OVOID, SEALANT, SLOFT and VISIONISER. The following may also prove useful: A51X, ZA7Q, XX2V and 53468279. But by far the largest problem with adventures lies in how to phrase certain things to produce the desired response. Below is another little list of some of the most useful phrases that can be used in certain rooms to your advantage. The room number is given in brackets.

PHRASES

Rotate dial (1), Give food (3), Pull cabinet (9), Change fuses/Repair drive with fuses (10), Examine drawer (14), Give apple to tortor (17), Type 53468279/Fire laser (23). Naturally there are a great number of other phrases that you must use. Next time I shall show you how to solve a few of the easier problems in the adventure - like how to get started without the whole place shutting down. I know from letters that I have received that the hole in location 15 has caused a number of problems. The solution to that is rather complicated and will be featured a bit later on. Until next month, have fun!

ATTRIBUTE 3 will relate to [COVER]
ATTRIBUTE 4 will relate to [SIZE]
ATTRIBUTE 5 will relate to [JARGON]
ATTRIBUTE 6 will relate to [SUBJECT MATTER]

The ADVISOR now asks you to CREATE VALUES for each ATTRIBUTE. At the prompt type in statements which describe the best, middle and least best qualities of each ATTRIBUTE. For our example file type the statements in the square brackets

ATTRIBUTE - COST
VALUE 1 will be [OVER £5]
VALUE 2 will be [£5]
VALUE 3 will be [UNDER £5]

ATTRIBUTE - PRINTING
VALUE 1 will be [LARGE PRINT]
VALUE 2 will be [AVERAGE PRINT]
VALUE 3 will be [SMALL PRINT]

ATTRIBUTE - COVER
VALUE 1 will be [HARDBACK]
VALUE 2 will be [PAPERBACK]
VALUE 3 will be [UNIMPORTANT]

ATTRIBUTE - SIZE
VALUE 1 will be [LARGE BOOK]
VALUE 2 will be [AVERAGE BOOK]
VALUE 3 will be [SMALL BOOK]

ATTRIBUTE - JARGON
VALUE 1 will be [SMALL AMOUNTS]
VALUE 2 will be [LARGE AMOUNTS]
VALUE 3 will be [NOT PRESENT]

ATTRIBUTE - SUBJECT MATTER
VALUE 1 will be [NOT TECHNICAL]
VALUE 2 will be [NOT TOO TECHNICAL]
VALUE 3 will be [VERY TECHNICAL]

You will see there are three grades of VALUE for every ATTRIBUTE, each could be desirable The most agreeable is put as the first VALUE, the second desirable as the second VALUE and the least agreeable as the third VALUE. At this stage it does not matter whether you have the VALUES for each ATTRIBUTE in the correct order - later you will learn how to RANK them for your own use.

STATEMENTS OF DECISIONS

When this set of data inputs have been completed press SPACE to continue. You will now have to think up some statements of decisions. Don't worry though, THE ADVISOR will guide you through the process with little difficulty using the existing data.

WHAT IE YOU HAVE ALL THESE?
1. Under £5
1. Large size
1. Hardback
1. No jargon
1. Not technical

type - BUY THE BOOK - AT ALL COSTS

IE YOU HAVE THESE!
2. £5
2. Medium size
2. Unimportant
2. Could be some jargon
2. Not too technical

type - CONSIDER BUYING THE BOOK

IE YOU HAVE THESE!
3. Over £5
3. Small print
3. Paperback
3. Could be lots
3. Very

type - LOOK FOR ANOTHER BOOK

These are the PRIMARY statements ie the first, middle and final. Now you will require the SECONDARY statements which fall between the others. What if VALUES to BUY THE BOOK AT ALL COSTS and CONSIDER BUYING THE BOOK are mixed? You would type something like - BUY AS SECOND BEST. IF the VALUES to CONSIDER BUYING THE BOOK and LOOK FOR ANOTHER are mixed, you would type something similar to - KEEP BOOK IN MIND. Having entered your last statement you will be returned to the menu.

VIEWING YOUR DATA

Type 2 to VIEW THE DATA and you will see the first set of ATTRIBUTES and their VALUES. Pressing SPACE lets you see the second set, (leads on to YOUR DECISIONS and back to the MENU).
You can correct any errors or change any data by selecting 5 CHANGE THE DATA. Each aspect will be presented and the program will wait for any change to be made.

RETURN keeps the data without change.
KEYBOARD INPUT changes the data.

Now make sure through VIEWing the data, that you have the VALUES placed in the order you wish. If not, THE ADVISOR allows you to make the necessary changes. For good guesswork this should be:

1. FIRST PRIORITY STATEMENT
2. SECOND PRIORITY STATEMENT
3. LAST PRIORITY STATEMENT

As an example, let us look at the ATTRIBUTE - COST

Suppose you decide that a primary cost factor for book purchase in UNDER £5 and that the last on your list would be a book for OVER £5. Select 9, RANK data and

you can make the changes necessary. BEWARE, because you have to go through ALL ATTRIBUTES and their VALUES.

EOR ATTRIBUTE COST
1. UNDER £5
2. £5
3. OVER £5
WHICH WOULD YOU PUT FIRST?

At the cursor following the question mark type 3 and press RETURN.

FOR ATTRIBUTE COST
1. UNDER £5
2. £5
3. OVER £5
WHICH WOULD YOU PUT SECOND?

At the cursor following the question mark type 2 and press RETURN

EOR ATTRIBUTE COST
1. UNDER £5
2. £5
3. OVER £5
WHICH WOULD YOU PUT THIRD?

At the cursor following the question mark type 1 and press RETURN

Continue through the program making changes or pressing RETURN until a request to type SPACE returns you to the main menu. As when using other software it is best to SAVE DATA as a sequential file before trying to perform analysis of the data.

THE NEXT STEP

Press 8 for DISK DIRECTORY and you will see there is a file called "BOOKS". If you try to write to this, corruption of other files could occur. Type 7 and the file will be SCRATCHed, if you are sure you want to do this. RETURN to the MENU and select 3 to SAVE THE DATA. You are given the current file name which can be used or another substituted. *** ALWAYS USE THIS METHOD WHEN YOU SAVE DATA ***

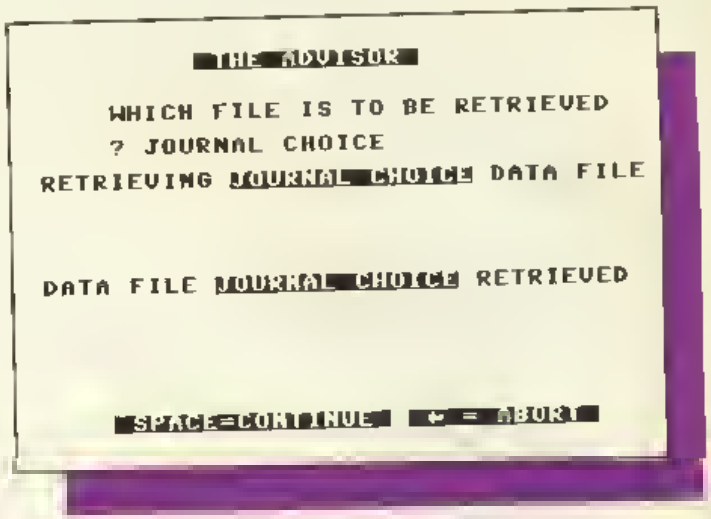
Now you can select option 6 and ANALYSE DATA. You will be shown a screen with the first ATTRIBUTE and its VALUES.

COST
1. UNDER £5
2. £5
3. OVER £5
TYPE NUMBER OE YOUR CHOICE

Say 1 and you will receive the screen stating that;

BOOKS SAYS
UNKNOWN EACTOR
BECAUSE:

COST IS UNDER £5
PRINTING IS UNKNOWN
COVER IS UNKNOWN
SIZE IS UNKNOWN
JARGON IS UNKNOWN
SUBJECT MATTER IS UNKNOWN



This is because all the data is not available for full analysis. Continue through the screens pressing "1" each time and the same comment will present itself. Only after selection of SUBJECT MATTER VALUE will a decision be offered.

BOOKS SAYS
BUY THE BOOK AT ALL COSTS
BECAUSE:

COST IS UNDER £5
PRINTING IS LARGE PRINT
COVER IS HARDBACK
SIZE IS LARGE BOOK
JARGON IS NON-EXISTENT
SUBJECT MATTER IS NOT TECHNICAL

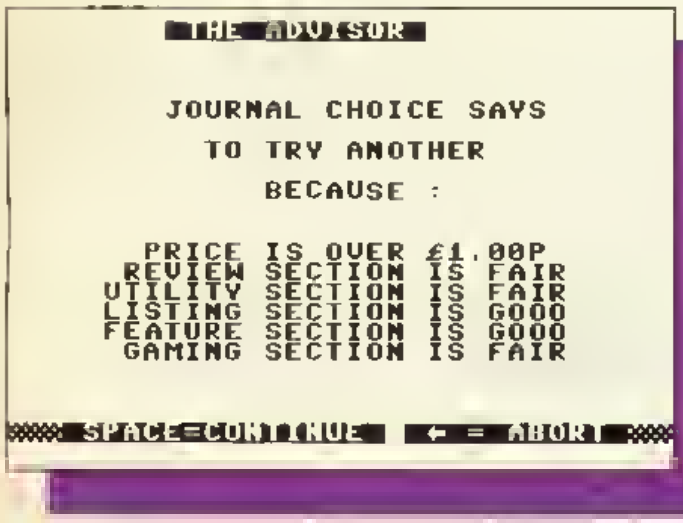
In the "real world" it would be difficult to find a set of VALUEs related to a purchase which meet all requirements. Always there is a "WHAT IF?..."

For example **WHAT IE?..**

COST IS OVER £5
PRINTING IS AVERAGE PRINT
COVER IS UNIMPORTANT
SIZE IS AVERAGE BOOK
JARGON IS NOT PRESENT
SUBJECT MATTER IS NOT TECHNICAL

The ADVISOR would say KEEP BOOK IN MIND.

Constantly changing your selection to ask THE ADVISOR "What if?" will show whether the decision to buy with various sets of VALUEs would be sensible or not When



you reach the desired decision or become tired of the whole thing press 0 and the program will be erased from the computers memory. Only by re-loading can you continue.

HOW IT WORKS

When inputting the selected VALUE by its number a process of totalling occurs. Obviously if all ATTRIBUTES are not being considered then a decision cannot be made. If all the top ranked VALUES are selected than a maximum score of 6 will be obtained. Equally should all low ranked attributes be selected a maximum score of 18 will result. A shared score between high and low rankings gives a tally of 12. Any score falling higher than the middle and lower than the maximum will produce the second decision, whilst those falling between the middle and lower produce a fourth decision.

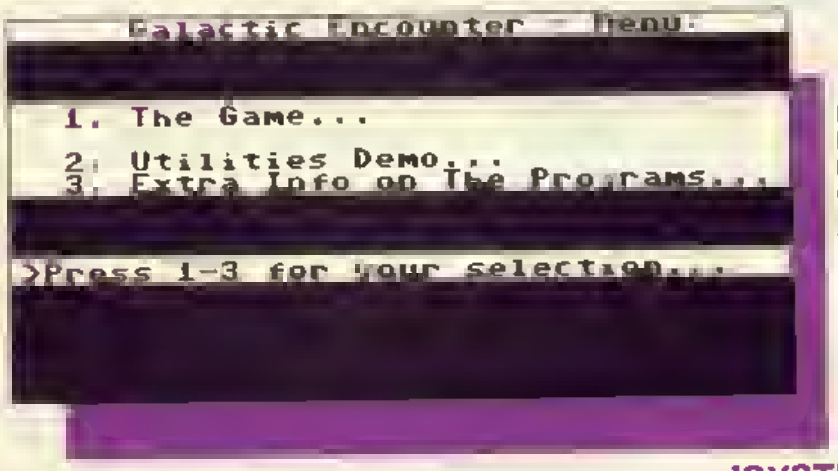
FUTURE DEVELOPMENT

The author cannot completely envisage the full potential of the program because of its endless possibilities. Perhaps there could be a transfer of information between program users so that all could benefit from the work being done by other interested users.

EDITORS NOTE:- Those readers that have in the past purchased our sister magazine YOUR COMMODORE, known now as YC, will no doubt be aware of the excellent REASONING ON THE C128 series that we published some time ago covering this fascinating subject of EXPERT SYSTEMS.

GALACTIC ENCOUNTER

Interplanetary war is the name of the game P. MAKEPEACE



The game is set in some distant galaxy where two squadrons of ships are doing battle on a 7 by 7 grid. The battle is played by two human opponents and the object is, not surprisingly, for each player to completely wipe-out the opponents ships.

SCENARIO

There are two classes of ship. The less powerful fighters, and the single destroyer. The fighter ships can move and fire only forwards and require rotating to change direction. Each player can have any number of these ships. The destroyer can move and fire in any direction (not diagonally) without having to rotate. This makes the destroyer very manoeuverable. In addition, the destroyer can shield up to five shots whereas the fighter can only sustain one.

Players move their ships in rounds of up to nine moves each. These moves van be used to move, rotate, or fire ships. After the nine moves, the round ends and the other player has their turn. This continues until either one of the players destroys all the other players ships. During a round, a player moving can end their turn at any time (see keys below).

When a ship is shot, it explodes and forms a cloud of ship debris. This cloud gets smaller and eventually dispersed altogether. The cloud shrinks every two rounds. Players ships cannot move through the debris but are able to shoot through it. Another obstacle is the blackhole which neither ships nor bullets can penetrate. There is no danger of being sucked into a blackhole.

CONTROLS

Either a joystick in Port 2 or the keyboard may be used at the same time. Keys are redefinable (see Setup). One player might choose to use the joystick whilst the other uses the keyboard, this would save joystick swapping. Players must agree not to wiggle joysticks or press keys during the others players turn! The keys are shown in brackets and are redefinable.

JOYSTICK KEYS ACTION

UP	[S]	Move ship one square forward
DOWN	[X]	Fire!
LEFT	[,]	Rotate ship anticlockwise
RIGHT	[/]	Rotate ship clockwise
FIRE	[A]	Select or deselect ship under cursor
SPACE/RET		End turn or shoot with the destroyer it selected.

(These cannot be redefined)

During the course of play, various messages will appear in the "STATUS REPORT" at the bottom right of the screen. To speed these up, press space or fire on the joystick. To temporarily halt the message, press any other key or joystick movement. The scroll speed can be adjusted (see Setup below).

SELECTION

When a round starts, a pulsing cursor appears below the first ship of the player moving. This can then be moved around with joystick/keys in the desired direction. When the cursor is positioned over a ship to be moved, press the select button (Fire or 'A'). The computer will issue an appropriate 'ding' sound and the cursor will change to a crosshair.

MOVEMENT

Once selected, the ship can be moved around as above.

When the player has finished moving, press the select button again and the cursor will revert to a square under the ship again.

FIRING

To fire, select the appropriate piece to do the firing and position it correctly. Press DOWN (NOT fire!!) or SPC/RETURN if the destroyer is selected. After confirming firing orders, press 1-9 for the number of shots to be fired, turns allowing. Following each shot, the result of the shot (hit, miss etc) will be displayed in the STATUS REPORT. AEter all the shots have been fired, the corresponding number of moves (and ships) are deducted, play continues as normal.

GAME SET-UP

This gives the players the opportunity to configure (modify) almost every aspect of the game. To use this, reply Y(es) or Fire on the joystick, to the prompt at the beginning of the game. A list will then appear showing the different things that can be modified including a LOAD/SAVE/DEFAULT option. A black bar will appear over the selected item, this can be moved up and down. When you wish to modify something, press Fire, the bar will change colour. Pressing LEFT or RIGHT will cycle through the different options. When you have finished, press Eire again and the bar will change to black again. To leave SET-UP press SPACE/RETURN To change the scroll speed, press UP (faster) or DOWN (slower) to view the different scroll speeds.

BDARD EDITING

Editing the board is done by selecting 'EDIT BOARD'. This then displays the number of ships for each player. The familiar pulsing square will appear which can be moved around as normal. To delete something, position the cursor over the piece and press Fire To create one, do the same. A blackhole will then appear and you can cycle through the different pieces by moving LEFT or RIGHT. You cannot have more than one destroyer! (the computer will automatically omit this option when there is one). When you are satisfied with the board, press SPACE/RETURN which will return to the SET-UP menu.
To LOAD or SAVE a board and the rest of the modifications, select 'LOAD SAVE DEF' and press "L" or "S" when prompted ("N" cancels this and returns to the SET-UP menu). Then press 0-9 for the particular file. The file will load or save over the old one. After loading, all the modifications will appear and the board will be redrawn. To return to the original board, press "D" (default) and allow a little wait as the variables are reset.

SUGGESTIONS

When editing the board try different colour schemes, the detault one looks fine on both colour and black and white but you may prefer otherwise. To add mystery, you might change the blackholes or players to the same colour as the background thus rendering them invisible. You might also try strategies with say one destroyer versus several fighters, or two destroyers against each other. On the disk there are several different files which you can try as well.
I cannot tell you much in the way of tactical tips but the big rule is to plan your move ahead, rather than try something and run out ot turns half way through. If you decide to sacrifice a destroyer with 3 hits left, make sure you hit more than 3 ships as it is much more manoeuverable (and useful) than a fighter Generally less moves requires more thought to determine what the other player is doing and act upon it. You may however, find one turn each a bit slow, it's up to you.

TECHNICAL DETAILS

For the more technical amongst you, we provide a breakdown of the game concept, programming techniques and some useful machine code information.

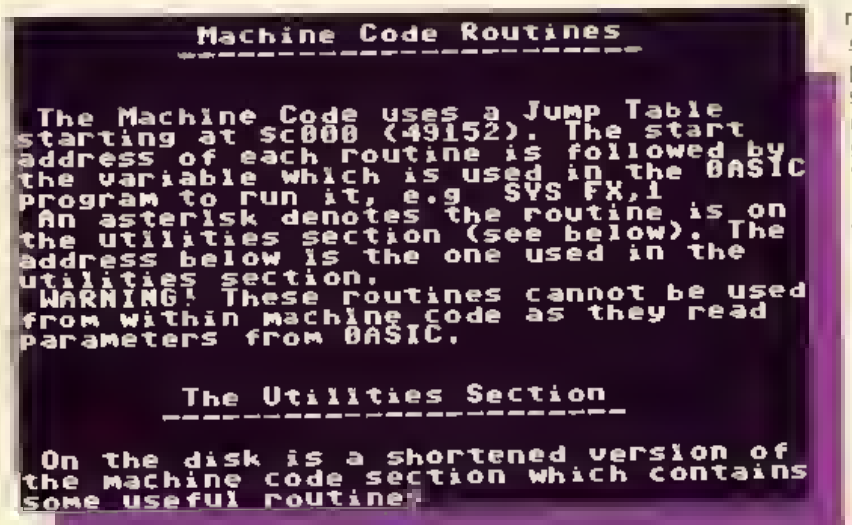
The entire program comprises four parts, the main BASIC program, a MACHINE CODE section, a EONT and the three SPRITES. On the disk is a file named "EXTRA INFO" (which can be listed to screen or printer) containing all the information concerning the MACHINE CODE. There is also some information about the main variables used in the program and the main routines, should you wish to investigate the program.

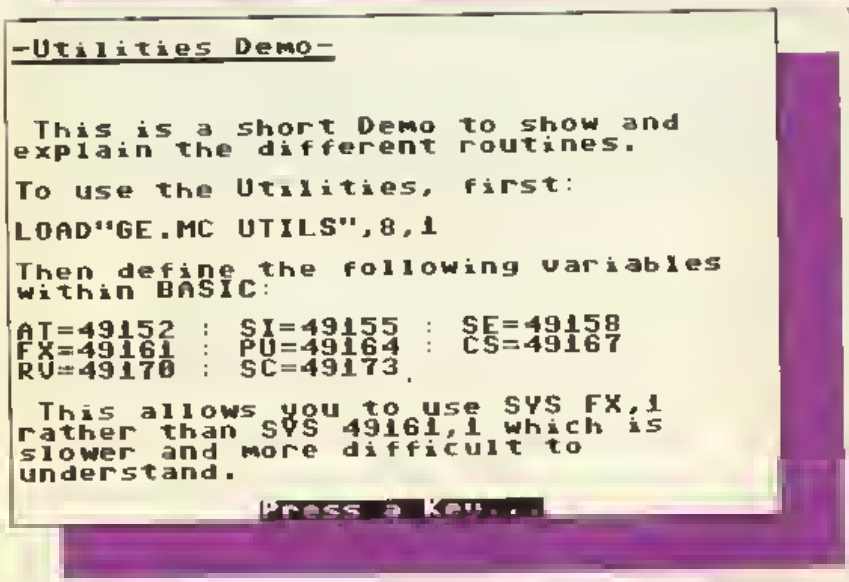
THE BASIC PROGRAM

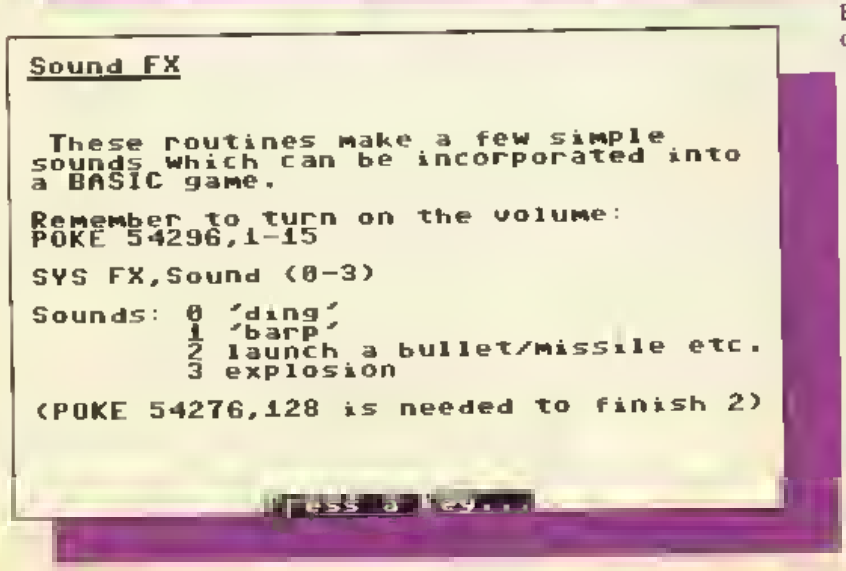
This starts in memory at $0801 (2049) which is the normal place for BASIC programs. It occupies 11K up to around $3300 (13056) which gives just over 1K for strings and variables which is not very much My upper limit is $3800 because of the font which occupies $3800-$3FE0, hence I do not have that much memory left. This forced me to transfer a lot of the BASIC routines to MACHINE CODE.
Initially when I started writing the BASIC program, it was almost entirely in BASIC, and it was appallingly slow, and occupied almost all of the memory I had (this was without the Set-up feature). Since then, I have optimised many routines within BASIC and converted other repetitive one into MACHINE CODE, such as the printing ship and board routines. Speed and playability were much improved.

THE MACHINE CDDE

This starts in the usual $C000 position and extends right up to $C5A0. In this space are all the routines which were previously done in BASIC and are now done faster than you can blink! The routines are accessed from BASIC from a JUMP TABLE starting at $C000. I have included on the disk an explanation of each of the







routines which can be printed to screen or printer (mentioned previously). Also on the disk is a selection of the routines from the main one which are not related specifically to the game such as PRINT AT, MESSAGE SCROLLING, PULSING SPRITE and some others which you may like to use in your own BASIC programs. To see how to use these, select "UTILITIES DEMO" from the GALACTIC ENCOUNTER menu.

THE FONT

This occupies memory from $7800 to $7FE0. It is a "COMPUTER" type font but also contains the board and border characters as well as the various ships in their different positions. Most characters remain as normal or "COMPUTERISED", the only major changes start after CHR$(192-255) which are the ships, blackholes and debris.
The font in the main game occupies $7800-$7FE0 which is in VIC chip bank 1 and so requires bank switching and moving the screen and sprite area. I was forced to have the font in memory here because I ran out of memory when the font was at $38090. If you like the font, there is an identical font file starting at $3800. It is very easily used and requires no bank switching. Simply type;

LOAD"GE.FONT $3800",8,1

POKE56,56 (This moves the top of Basic to $3800 so no strings overwrite the font).

POKE53272,31 (This points the VIC chip to look in the right place for the font).

THE SPRITES

This is the smallest part starting at $0340 and finishing at $03FF, just before the screen starts. Using the tape or a reset button will destroy these sprites. The sprites are the blank square, bullet and crosshair in that order and can be accessed by POKEing 2040 with 13, 14 or 15 after turning on the sprite and putting it on the screen.
I hope you enjoy the game!!

EXPLAINING INTERRUPT REQUESTS

Programming tips for all ALEX BLEWITT

In this article I will explain what MULTI-TASKING and the INTERRUPT REQUESTS are on the C64. I have also included on this disk a few example programs which use IRQ's to create simple but useful effects. This article is aimed at programmers who know a little machine code, but do not give up and turn the page if you know nothing about machine code! If you follow through the examples given, you should be able to understand them and adapt them to suit your own needs.

WHAT IS MULTI-TASKING?

Multi-tasking is when a computer is simultaneously running two programs. This can be quite useful in game-writing, as one program can be used to play the music whilst the other can be used to display the graphics. But where does this fit in with our trusty friend, the 64? Well, the computer itself cannot run two programs at once, but it can get reasonably close. The system has IRQ's (Interrupt ReQuests) which stop all activity that is running, and perform "Housekeeping" (e.g.read the keyboard, update the clock etc). You haven't noticed this before, because it does it so fast. In fact, it does an IRQ every 'JIFFY', or 1/50 th of a second to the less technical people.

WHERE IS THIS HOUSEKEEPING CODE?

The code itself is in ROM, at address 59953 (or $EA31). In order to perform this code, the computer has to know where it lies in the ROM. You may wonder why I am telling you this, as the code is in ROM and therefore cannot be changed, nor added to. The answer lies in the fact that the JUMP address is stored as a VECTOR address in RAM (where it can be changed). This address is found in locations 788 and 789 ($0314 & $0315). 788 contains the LOW BYTE of the address and 789 contains the HIGH BYTE. To find the actual address of the IRQ code, use the following formula:

'PRINT (PEEK(788)+PEEK(789)*256)'

This will give an address of 59953 ($EA31). We will need to go to this address later, to go to at the end of our routine. If this is not done, the system will crash! So how can you run two programs at once? The answer is: add the second program to the 'HOUSEKEEPING' code. This program will then be executed every 1/50th of a second.

SETTING UP INTERRUPTS

To set up an interrupt program, we have to change the housekeeping address values of 788 and 789 to point to the new program. The following program is situated at 49152 ($C000) and sets the interrupts to 49184 ($C020). The interrupt program ($C020) onwards doesn't do anything, but it shows you how to set up the interrupts using a machine code program. (See general setting up program)

GENERAL SETTING UP PROGRAM

```
10      *=$c000         ;START AT 49152
20      SEI             ;DISABLES THE
30                      ;INTERRUPTS WHILST
40                      ;SETTING UP
50      LDA #$20        ;LOW BYTE ADDRESS
60      STA $0314       ;STORE IN LOCATION 788
70      LDA #$C0        ;HIGH BYTE ADDRESS
80      STA $0315       ;STORE IN LOCATION 789
90      CLI             ;RESTORE INTERRUPTS
100     RTS             ;AND GO BACK TO BASIC
110     *=$C020         ;INTERRUPT PROGRAM
120                     ;ADD CODE HERE....
190     JMP $EA31       ;LAST INSTRUCTION MUST
GOTO STANDARD HOUSEKEEPING CODE
```

All the examples on the disk are given as three types of files.

1) Basic demonstration programs, suffix .PRG Load these like ordinary basic programs.
2) Machine code files.
3) The source version that will need to be assembled with an assembler.

To load the machine code files, type
'LOAD"FILE.MAC",8,1', and start them with 'SYS 49152'. To load the source listings, type 'LOAD"FILE.LIST",8' or type them in (but loading them is a lot easier!) for an assembler (such as Dave Weaver's 6510+ ASSEMBLER. CDU May/June 89).
To run these, type
'ASSEMBLE (return) SYS 49152 (return)'. All these demos can be loaded from the interrupts menu, which can be loaded from the CDU menu on the disk, or by typing LOAD"INTERRUPTS",8 (ret) RUN (ret).

THE TURN OFF

The interrupts must be disabled before any I/O is attempted, or you will see a different effect than you want! In order to turn them off, type;

POKE 671,49:POKE 672,234:SYS 64659 (ret)

The address used in the little set-up program is $C000, mainly because it is the most convenient place in memory to write code. It doesn't have to be at $C000 and the set-up code does not have to be anywhere near the interrupt code, but the set-up code must contain the address of the interrupt code. If you have no knowledge of machine code, then don't bother to try to move the code. $C000 is an easy address and is easily called from and used with BASIC. The following examples will reside at this memory location.

THE DEMOS ON THE DISK

<KEY BORDER>
FILE "KEYPRESS.LIST"

```
10      ;"@:KEYPRESS.LIST"
20      *=$C000
30       SEI
40      LDA #$20
50      STA $0314
60      LDA #$C0
70      STA $031S
B0      CLI
90      RTS
100     *=$C020
110     LDA $C5         ;GET KEY PRESSED
120     CMP #$40        ;IS IT NO KEY?
130     BEQ QUIT        ;IE IT IS, LEAVE
140     STA $D020       ;PLACE KEY VALUE IN
150                     ;BORDER COL
160     QUIT            ;QUIT
170      JMP $EA31      ;HOUSEKEEPING
```

This first short routine (see KEYPRESS.LIST) serves no actual purpose, but shows how a machine code routine can run whilst a basic program is running. The demonstration program shows this (File "KEYPRESS.PRG").

The program. Line 10 holds the name of the program (which is used by the 6510+) Lines 20-90 set up the interrupts to point to the program. Line 100 tells the compiler to start at $C020. Line 110 LoaDs the Accumulator with the value of $C5. This is the value of the last key pressed. Line 120 checks to see if it is no key (value $40), and if no key has been pressed, it goes to 'QUIT'. Otherwise, it pokes the value (ie STores the Accumulator) into $D020 or 53280 - the border colour location. The program then returns to the Housekeeping by having the last instruction to JMP $EA31.

<F-KEYS UTILITY>
FILE "F-KEYS.LIST"

```
10                      ;"@:E-KEYS.LIST"
20      *=$C000
30       LDA $0314      ;HAVE INTERRUPTS
40      CMP #$20        ;ALREADY BEEN PUT IN?
50      BEQ KEYPROG     ;IE SO, GOTO KEYPROG
60      SEI
70      LDA #$20
B0      STA $0314
90      LDA #$C0
100     STA $031S
110     CLI
120     RTS
130     *=$C020
140     LDA $CS         ;GET KEYPRESS
150     CMP #$04        ;IS IT E1?
160     BEQ E1          ;GOTO E1
170     CMP #$05        ;IS IT F3?
180     BEQ E3          ;GOTO F3
190     CMP #$06        ;IS IT FS?
200     BEQ F5          ;GOTO FS
210     CMP #$03        ;IS IT F7?
220     BEQ E7          ;GOTO F7
230     JMP $EA31       ;HOUSEKEEPING
240 F1                  ;E1
2S0     LDA #$00        ;SET BASE TO 00
260      JMP SHIFT      ;TEST SHIFT
270 F3                  ;E3
2B0     LDA #$10        ;SET BASE TO 10
290     JMP SHIFT       ;TEST SHIFT
300 FS                  ;ES
310     LDA #$20        ;SET BASE TO 20
320     JMP SHIFT       ;TEST SHIFT
330 F7                  ;F7
340     LDA #$30        ;SET BASE TO 30
350      JMP SHIET      ;TEST SHIET
360 SHIFT               ;TEST SHIFT ROUTINE
370     LDX $028E       ;GET SHIFT PATTERN
3B0     CPX #$00        ;NO SHIFT
390     BEQ OUTPUT      ;GOTO OUTPUT
400     CPX #$01        ;SHIET KEY
410     BNE CBM         ;NO, GOTO CBM
420     ADC #$07        ;ADD 7 TO BASE
430      JMP OUTPUT     ;GOTO OUTPUT
440 CBM                 ;COMMODORE TEST
450     CPX #$02        ;IS IT C=?
460     BNE CTRL        ;NO, GOTO CTRL
470     ADC #$3E        ;ADD 3F TO BASE
4B0     JMP OUTPUT      ;GOTO OUTPUT
490 CTRL                ;CONTROL TEST
S00     CPX #$04        ;IS IT CONTROL
S10     BNE OUTPUT      ;GOTO OUTPUT
520     ADC #$47        ;ADD 47 TO BASE
S30 OUTPUT              ;OUTPUT KEY ROUTINE
540     STA TEMP+1      ;PUT ADDRESS IN LDA
SS0     LDY #$00        ;SET COUNTER TO 0
S60 TEMP                ;POINTER
570     LDA $C100,Y     ;GET CHAR
SB0     STA $0277,Y     ;AND PUT IN BUEEER
590     1NY             ;NEXT CHAR
600     CPY #$08        ;ALL B DONE?
610     BNE TEMP        ;NO, GOTO TEMP
```

```
620     STY $C6          ;BUEEER EULL SIGN
630     LDY #$00         ;COUNTER TO 00
640THINGY                ;WEIRD NAMED ELAG?!
650     JSR $EEB3        ;DELAY 1MS
660     DEY              ;DO IT 256 TIMES
670     8NE THINGY       ;GO 8ACK TO THINGY
680     JMP $EA31        ;HOUSEKEEPING
690 KEYPROG              ;PROGRAM F-KEY
700     JSR $AEED        ;GO PAST COMMA
710     JSR $879E        ;GET NUM8ER
720     CPX #$00         ;IS IT 0?
730     8EQ ERROR        ;ERROR!
740     CPX #$11         ;MORE THAN 17?
750     8CC GETKEY       ;NO, OKAY
760ERROR                 ;ERROR ROUTINE
770     LDX #$0E         ;ILLEGAL QUANTITY
780     JMP $A437        ;PRINT ERRROR
790     GETKEY DEX       ;TAKE ONE AWAY
800     STX $FD          ;BUNG IT IN $FD
810     TXA              ;AND IN ACC.
820     CLC              ;CLEAR C FLAG
830     ROL A            ;MULT 2  (2)
840     ROL A            ;MULT 2  (4)
850     ROL A            ;MULT 2  (8)
860  STA ADDRESS+1       ;POKE INTO LDA
870     JSR $AEED        ;GO PAST COMMA
880     JSR $AD9E        ;GET STRING
890     STA $FE          ;STORE LENGTH
900     JSR $86A3        ;DISCARD STRING
910     LDY #$00         ;COUNTER TO 00
920GETMORE               ;GETMORE
930     LDA ($22),Y      ;GET STRING
940     CPY $FE          ;END OE STRING?
950     8CC ADDRESS      ;NO, GOTO ADDRESS
960     LDA #$00         ;SET CHAR TO 00
970 ADDRFSS              ;POINTER
980     STA $C100,Y      ;PUT IN MEMORY
990     INY              ;ADD 1 TO COUNT
1000    CPY #$08         ;8 CHARS?
1010    BNE GETMORE      ;NO, GOTO GETMORE
1020    RTS              ;RETURN
```

The second demo, F-KEYS is a rather more complex routine. The program checks whether the interrupts have already been set up, and if they have, it jumps to 'KEYPROG', where a key is programmed. There is not enough space to go through the entire program, so I will just explain how the interrupts are used.

Lines 60-120 are our friendly set up routine. The interrupt program begins at line 140 (at $C020). Here, the program gets the value of the last key pressed, and lines 150 to 220 check to see if any of the F-keys have been pressed. (Some of the more observant of you will have noticed that there are no references to F2,F4,F6 or F8 - this is because the value of $C5 is not affected by the shift key; it will be the same value whether the shift key, commodore key or control key is pressed.) If a match is not found (ie if no f-key has been pressed), then the program goes to $EA31 - the Housekeeping. If a match is found, the program sets an address base value for the location of the f-key string. The shift key is then checked by getting the value of $028E. This indicates which shift has been pressed. If it is SHIFT, the value is 1. If COMMODORE is pressed, the value is 2. If CONTROL is pressed, the value is 4. The rest of the program gets a bit complicated, but ends up at line 860 with our Housekeeping JuMP.

INTERESTING PROGRAMMING POINTS

1) The value of the shift key is in memory location $028E. (1=SHFT,2=C=,4=CTRL), thus providing another 8 F-keys.

2) JSR $AEFD - go past comma. This is part of the procedure to program the F-key. It must be used to go past the comma in the SYS 49152,1,"text" line.

3) JSR $B79F - get a number. This gets the number and stores it in the X-reg. 4) JSR $A437 - print error. The value of the X-reg produces different error messages. ($0E = illegal quantity error)

5) JSR $AD9E - get string length. Length returned in X-reg.

6) JSR $B6A3 - discard string. This also puts the string pointer in $22, so the string can be read in by LDA ($22),Y or a similar routine.

By using a call to $AEFD and then to read either a variable or string, you can pass on variables to a machine code program. I hope you find this useful.

OPERATION OF THE DEMO

See F-KEYS.PRG or read the next paragraph

```
SYS 49152          - set up interrupts.
SYS 49152,key#,"text" - program key number with text.
                    key# can be anything from 1 to 16.
                    text is the string to be placed.
                    if you wish to have a return after
                    it, add +CHR$(13) to the end.
F-keys              - will move the string into the buffer,
                    so works both in direct + program mode.
```

Corresponding shift keys

F1,F3,E5,F7	**- no shift.**
F2,E4,E6,F8	**- shift.**
F9,F11,F13,F15	**- commodore.**
F10,E12,E14,E16	**- control.**

As usual, to turn off the interrupts, the line is;
POKE 671,49:POKE 672,234:SYS 64659

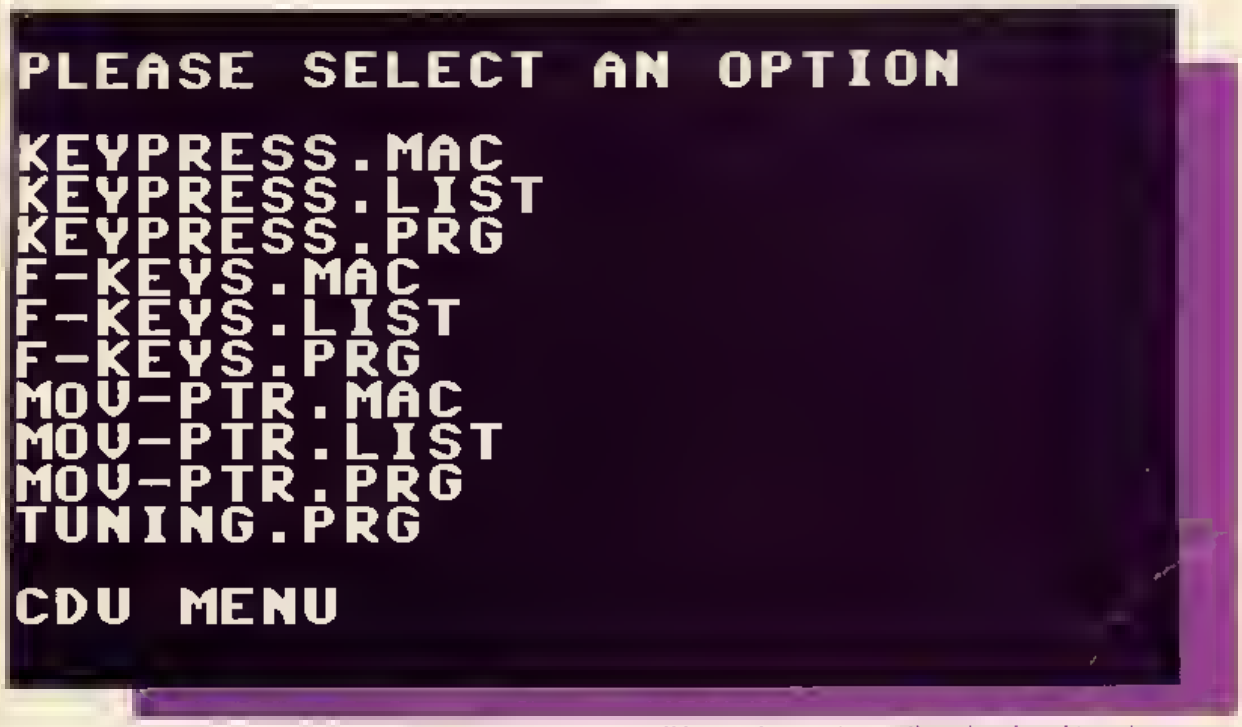
<POINTER MOVER>
EILE"MOV-PTR.LIST"

```
10                  ;"@:MOV-PTR.LIST"
20      *=$C000
30      SEI         ;SET UP INTERRUPTS
40      LDA #$20
50      STA $0314
60      LDA #$C0
70      STA $0315
80      CLI
```

```
90      LDA $C01C       ;GET FINE X-VAL
100     STA $C0BD       ;POKE INTO PROGRAM
110     STA $C0CD       ;IN TWO PLACES
120     LDA $C01D       ;GET FINE Y-VAL
130     STA $C0DA       ;POKE INTO PROGRAM
140     RTS             ;LEAVE
150     BYT $10         ;FINE X-VAL
160     BYT $29         ;FINE Y-VAL
170     *=$C020
180     LDA $DC00       ;GET JOY POS PORT 2
190     STA $C01E       ;PUT IT IN MEMORY
200     LSR $C01F       ;SHIFT RIGHT
210     BCS D           ;IF BIT 0, NEXT BIT
220     JSR UP          ;OTHERWISE UP
230 D   LSR $C01E       ;SHIFT RIGHT
240     BCS L           ;IF BIT 0, NEXT BIT
250     JSR DOWN        ;OTHERWISE DOWN
260 L   LSR $C01E       ;SHIFT RIGHT
270     BCS R           ;IF BIT 0, NEXT BIT
280     JSR LEFT        ;OTHERWISE LEFT
290 R   LSR $C01E       ;SHIFT RIGHT
300     BCS F           ;IF BIT 0, NEXT BIT
310     JSR RIGHT       ;OTHERWISE RIGHT
320 F   LSR $C01E       ;SHIFT RIGHT
330     BCS Q           ;IF BIT 0, NEXT BIT
340     JSR FIRE        ;OTHERWISE FIRE
350 Q   JMP $FA31       ;HOUSEKEEPING CODE
360 UP                  ;MOVE UP
370     LDY $D001       ;GET Y-POS
380     CPY #$1F        ;CHECK IF NOT OFF SCRN
390     BEQ QUITU       ;IF OFF, LEAVE
400     DEC $D001       ;MOVE UP ONE
410 QUITU               ;LEAVE
420     RTS
430 DOWN                ;MOVE DOWN
440     LDA $D001       ;GET Y-POS
450     CMP #$F9        ;CHECK IF NOT OFF SCRN
460     BEQ QUITD       ;IF OFF, LEAVE
470     INC $D001       ;MOVE DOWN ONE
480 QUITD               ;LEAVE
490     RTS
500 LEFT                ;MOVE LEFT
510     LDA $D010       ;GET X-MSB
520     AND #$01        ;CHECK WHICH SIDE
530     CMP #$01        ;CHECK MSB RIGHT
540     BNE ONLEFT      ;NO, GOTO ONLEFT
550     DEC $D000       ;MOVE LEFT
560     BMI MSB0        ;CHECK IF PASSED LINE
570     RTS             ;LEAVE
580 MSB0                ;0>MSB
590     LDA $D010       ;GET REG
600     AND #$FE        ;0>BIT0
610     STA $D010       ;PUT IT BACK
620     RTS             ;LEAVE
630 ONLEFT              ;ONLEFT
640     LDX $D000       ;GET X-POS
650     CPX #$01        ;CHECK IF NOT OFF SCRN
660     BEQ QUITL       ;IF OFF, LEAVE
670     DEC $D000       ;MOVE LEFT
680     QUITL           ;LEAVE
690     RTS
700     RIGHT           ;MOVE RIGHT
710     LDA $D010       ;GET X-MSB
720     AND #$01        ;CHECK WHICH SIDE
730     CMP #$01        ;ON RIGHT?
740     BEQ ONRIGHT     ;YES, GOTO ONRIGHT
750     INC $D000       ;MOVE RIGHT
760     LDA $D000       ;GET X-POS
770     CMP #$00        ;CHECK IF PASSED LINE
780     BEQ MSB1        ;IF HAS, GOTO MSB1
790     RTS             ;LEAVE
800 MSB1                ;1>MSB
810     LDA $D010       ;GET REG
820     ORA #$01        ;1>BIT0
830     STA $D010       ;PUT IT BACK
840     RTS             ;LEAVE
850 ONRIGHT             ;ONRIGHT
860     LDA $D000       ;GET X-POS
870     CMP #$57        ;CHECK IF NOT OFF SCRN
880     BEQ QUITR       ;IF OFF, LEAVE
890     INC $D000       ;MOVE RIGHT
900 QUITR               ;LEAVE
910     RTS
920 FIRE                ;FIRE ROUTINE
930     LDA $D010       ;GET MSB
940     AND #$01        ;GET BIT
950     BNE FIREMSB     ;IF <>0, GOTO FIREMSB
960     LDA $D000       ;GET X-POS
970     SBC #$10        ;TAKE AWAY FINE
980     LSR A           ;DIV 2
990     LSR A           ;DIV 4
1000    LSR A           ;DIV 8
1010    STA $C01A       ;STORE X-POS
1020    JMP YBIT        ;CALCULATE Y-POS
1030 FIREMSB            ;FIREMSB
1040    LDA $D000       ;GET X-POS
1050    ADC #$14        ;ADD 16 TO MAKE FINE
1060    SBC #$10        ;TAKE AWAY FINE
1070    LSR A           ;DIV 2
1080    LSR A           ;DIV 4
1090    LSR A           ;DIV 8
1100    ADC #$1D        ;ADD 29
1110    STA $C0F0       ;STORE X-POS
1120 YBIT               ;Y CALCULATE
1130    LDA $D001       ;GET Y-POS
1140    SBC #$29        ;TAKE AWAY FINE
1150    LSR A           ;DIV 2
1160    LSR A           ;DIV 4
1170    LSR A           ;DIV 8
1180    STA $C0F1       ;STORE Y-POS
1190    RTS             ;LEAVE
```

The last demo that I have written is quite long and there is not enough space to describe the whole program in this article. However, you will find it on the disk as "MOV-PTR.LIST" or "MOV-PTR.MAC" The comments by the side of the program should help you understand it. I will, however go through the part of the program which utilises the interrupts.

Lines 30-80 are the normal set up routines. The next few lines move the fine tuning values into the right parts of the program. The interrupt program starts at line 180. The program monitors the joystick at $DC00, and if it finds a bit set it will jump to a routine, depending on which bit has been set. The routines are UP, DOWN, LEFT, RIGHT and FIRE (not necessarily in that order!). You can go through the routines in the listing. They move Sprite 0 around.



OPERATION OF THE PROGRAM

Once the program has loaded and started running (by SYS 49152), then the sprite needs to be set up. The routine is fairly simple. Sprite 0 is moved around the screen by the joystick in port 2. When FIRE is pressed, the current character coordinates are stored in memory locations $C0F0 and $C0F1 (or 49392 and 49393) for the x-col and y-row values. There is a basic demo of this on the disk called "MOV-PTR.PRG" which loads and runs the machine code. The sprite can then be moved atound, placing '*' when you ptess the fire button. The routine is also used by the menu "INTERRUPTS".

A feature of this pointer program is the 'Fine tuning'. I reasoned; the tip of the pointer is not always at the top left of the sprite, so something must be done to account for it. Enter the fine tuning value. The fine tuning value gives whete the centre of the pointet is. To calculate the X-value, the equation is X=16+no bits across sprite, and the Y-value equation is Y=40+no bits down sprite. Alternatively, use the tuning program on the disk to calculate it for you. Set up your sprite, as you would normally, and then load and run "TUNING.PRG". Then move the pointer to the centre of the ted circle and press fire. The computer will then give the X-value and Y-value. To install these fine tuning values in your own program, and to load the machine code file, use the following lines:

```
10 A=A+1:IF A=1 THEN LOAD "MOV-PTR.MAC",8,1
20 POKE 49180,(X-VALUE)
30 POKE 49181,(Y-VALUE)
40 SYS 49152: REM TURN ON INTERRUPTS
```

This will install the interrupts and the fine tuning values will be used. N.B. they will not be placed into the program directly; they will only be installed when you type 'SYS 49152'. You can do this even if it is still running, though. If you poke these values into the locations, and then save the program using a monitor, then these values will be preserved and used every time you call the routine. (The fine tuning values are found in line 150 and 160 of the assembler source code listing). You are free to copy this routine and use it in your own programs.

Well, that about wraps it up for my article then. I hope that you have found it interesting, and I also hope you have been able to understand what I have been blithering on about. One last point; in your interrupt program, don't have an everlasting loop (ie. no '10 GOTO 10's). The program must be called so that it will finish and then do the Housekeeping, otherwise the computer will crash!

To sum up, here is a list of the programs on the disk and what they do:

"INTERRUPTS" :Menu to load these programs
"KEYPRESS.LIST" :Assembler listing of Keypress.mac
"KEYPRESS.MAC" :Colour border program
"KEYPRESS.PRG" :Basic demo utilising Keypress.mac
"F-KEYS.LIST" :Assembler listing of F-keys.mac"
"F-KEYS.MAC" :F-keys program
"F-KEYS.PRG" :Basic demo utilising F-keys.mac
"MOV-PTR.LIST" :Assembler listing of Mov-ptr.mac
"MOV-PTR.MAC" :Sprite mover program
"MOV-PTR.PRG" :Basic demo utilising Mov-ptr.mac
"TUNING.PRG" :Utility to find tuning values for Mov-ptr

Have fun and enjoy!!

TECHNO-INFO

This is the FIRST ANNIVERSARY!! of Techno Info JASON FINCH

Well folks, this is just a little opening comment from the Tech. Dept. at CDU. You may or may not realise that TECHNO INFO is one year old today and so celebrations are in order methinks. Twelve months and hundreds of letters later I would like to thank all you readers who have sent me those letters, without which this section would be no more. And you wouldn't want that, would you? Well, I certainly wouldn't! Although I try to answer every letter, please understand it is not always possible for me to help everyone. To those few that never received replies of any sort, and there shouldn't be many, I apologise here and now.

On a better note though, hopefully, I would like you to write to me at the special TECHNO INFO address if you have any comments to make about this section of the best serious magazine for C64 and C128 users. What do you think of the service? Could it be improved in any way, or should it be altered somehow? If so, what changes would you like to see? I want to continue to make TECHNO INFO as useful and helpful as possible and I will value your comments. As an incentive and an alternative to champagne, I vow to send three packs of ten disks to the first three readers that tell me their thoughts, whether favourable or otherwise. Meanwhile, don't forget to keep sending in details of programming or other software problems or just general queries that you would like answered. And also keep those popular tips flooding in.

This month we have a truly international section, with letters from people in countries as far afield as the NETHERLANDS and GERMANY to SAUDI ARABIA and AUSTRALIA. So, without further ado, let's start the proceedings.

KEYBOARD SCANNING

Dear CDU,
I am in the process of writing a program that requires a method for differentiating between which SHIFT key has been pressed. It is a sort of bat and ball game and I wanted to make the keypresses a bit different - most people tend to go for alphabetic keys when directional control is from the keyboard. But I have come up against a brick wall. I thought that location 653 might have been able to help but from that I can only detect whether a SHIFT key has been pressed, not which one. I know very little machine code and so would be grateful of a BASIC routine or explanation as to how to go about the task. I look forward to your help.
Jochen Kaufmann, Plymouth.

Dear Jochen,
The secret to what you want to do lies with the CIA near the top of memory. The locations that are most important are 56320 and 56321, the data ports. You can detect which key has been pressed by POKEing 56320 with a suitable value and then by PEEKing location 56321. This value will then tell you which key or keys has been pressed. Before you do that from BASIC you must stop one of the internal timers to get an accurate reading. This is done by setting all the bits in location 56334 to zero. In simple terms, that just means you enter POKE 56334,0. The process required to read any key is too complex to explain fully here but it relies on a sort of matrix layout of the keys and the clearing and setting of various single bits controls which column is read and then which row. Suffice it to say that to detect the LEFT SHIFT key (also SHIFT LOCK) you should POKE 56320,253 and then check that PEEK(56321) = 127. If it does then LEFT SHIFT has been pressed. For RIGHT SHIET you POKE 56320,191 and then check 56321 for the number 239. I hope you find this information useful. Tune in again next month and I shall provide for you a routine on the disk to illustrate the point. Unfortunately this month there is not sufficient room for the program I wanted.

PRINTING 'OUT'

Dear CDU,
Whenever I attempt to dump the machine code from the 6510+ Assembler to my Citizen 120D printer nothing happens - the coded bytes and the instructions are just listed to the screen as usual. I have used the commands OPEN 1,4: CMD 1 before assembling the code both linked by colons and on separate lines but to no avail.

The list is just given from the position of the OUT command onwards on the monitor. I write to you in despair as I cannot think of any other commands that could be issued to dump it to the printer. Surely it should be possible to print the code for future reference. Could you tell me whether there are any alterations that I could make to the program to enable it to work as I want, or are there some other commands that I have overlooked?
Michael Powell, Nottingham.

Dear Michael,
Although it would seem that OPENing a channel for the printer would do something, in this case, as you rightly point out, it doesn't. And that is because as soon as you execute the ASSEMBLE command all open files and channels are closed by a jump to the Kernal ROM. You can prevent this from happening by diverting the start of the routine to after the jump has been done. This you do by entering POKE 31335,154 before you enter the OPEN 1,4. CMD 1 line. If you do that then it should start printing after the third pass as you expected.

WAGE CALCULATIONS

Dear CDU,
For the past two years I have run quite a reasonably sized business, employing around ten people depending upon the demand of our product, snail breeding kits. However, I have become aware that a number of the employees are acquiring rather flippant attitudes, taking advantage of the high wages that they are paid. Some just think that they can swan off on holiday whenever the fancy takes them, and they have no concern for the already damaged snail population. One has just recently returned from a holiday in Germany which he took during the busiest period of the year when snail activity is high. The accounts for the business are run on a C128 computer with the help of the GEOS package which I use because of its excellent spreadsheet and database - a record of all our customers is needed so that we can send out circulars, another costly process. I need a program that will show me on the screen the balance between all the different aspects of running the business, and hopefully this will enable me to build up a set of wage increases in line with inflation and the higher cost of living. At the moment the advertising manager is on a five figure wage and I think that this needs to be revised although I would like to see everything together so that I know precisely what I can offer him. Could you please inform me of a company that sells such software so that I may be able to cut down the rate of wage increases slightly. I need to show the employees that wage increases need to be minimised due to output restrictions. With the saving of money in that field we may be able to improve the packaging and add something extra to the kits - perhaps a third snail - to make them more attractive for the customer, thereby allowing the reduced snail population to increase a bit faster. I hope you can help.
Paulette "Save the Snail" Yves, Exeter.

Dear Paulette,
Unfortunately I am unaware of a program other than those in the GEOS package that will help you to see all your expenses and so on lined up together. You could try the Graph-Ed program that appeared a little while back to assess the costs incurred by your company but I do not know of a program that will calculate exactly what sort of wage increases would be required. As I am sure you are aware they will have to be balanced with demand of your product but your company will still need to make a profit. As I say, try out some graph programs like the CDU one or the GeoChart package. I wish you luck in your search and hope that the snail population picks up soon.

GeoPROBLEMS

Dear CDU,
Being 'down under' and a little 'out back' I find it difficult and time consuming trying to communicate promptly on various subjects that crop up. I am using the GEOS system extensively in business and private and find the packages outstanding. I recently, four months ago, sent an order to Berkeley Software in California. I have sent two letters to their customer service with no replies to date. Is there anyone, CDU reader or GEOS user, with the same or similar communication problem with the company in the USA. You see, I do not know whether the company are still at the same address, Shattuck Avenue. I am aware that FSSL are the agents for GEOS software in the UK but I do not have the address. Could you tell me what it is? I do hope you can find space to print my letter, and if they read CDU in California, so much the better!!
George Wynne, Australia.

Dear George,
I unfortunately do not know whether or not Berkeley Software is still at the address you state but I can give you the address of FSSL and ask if anyone else has had problems contacting the company by post. It may be that they have changed their address. FSSL are an excellent company and will probably be able to give you the address or find out what is happening if you supply proof that the goods are paid for. FSSL's address is Masons Ryde, Defford Road, Pershore, Worcestershire, WR10 1AZ. Good luck

APPLYING CODE

Dear CDU,
I have a 64C and have been a novice for three years now as far as machine code goes. Here in Saudi I have found it very difficult to find any books or magazines on "how to do" machine code or assembly language. There are plenty of books about the instruction sets and what assembly language is, but never the application of it.

What I would like to see in your mag is a step by step guide on how to use assembly language. Who knows, one day I might be able to write a game called Desert Shield or something.
N.Sirett, Saudi Arabia.

Dear Mr.Sirett,
I don't really think that there is a lot that can be said on applying machine code. It all depends upon what you want the code to do and no two situations are likely to be tailored to the use of the same piece of code. What I would recommend is that instead of buying books, play about yourself with other peoples programs and see what they do. Start out by writing your own very short routine just to display a letter somewhere on the screen. You do that by first loading the accumulator with a value and then storing it at a location on the screen, followed by an RTS instruction. Applying code relies on your understanding of how the A,X and Y registers can be used basically, and to what uses they can be put in accomplishing the task you want. In a game you may want a routine to read in whether or not the fire button is pressed on a joystick and if so to fire a bullet. You need to play about with locations in memory, reading them in with the LD? commands and comparing them with other values and so on. I don't think that it would be possible to write an awful lot on the application of assembly language as it is too specialised and directed towards certain tasks. I hope you get the hang of it eventually - just keep playing about and experimenting with the instructions.

SOUND EXPANDER

Dear CDU,
I am writing to you as I own Commodore's Sound Expander for the C64. I would like to know how I can use it in my own programs, but unfortunately the manual gives no hints as to how to use it without the additional software supplied. I would be very grateful of any help.
M.Wyld, Spalding.

Dear Mr.Wyld,
This is one of those times when I cannot offer any help whatsoever because I am not familiar with the piece of software that you mention. However, instead of just ignoring your plea I am publishing your letter in the hope that someone out there has got this package and knows how to use it in the way that you want. If anyone has got any info, please do send it to us.

128 ROUTINES

Dear CDU,
First of all, the program "Picture Print" in your February 1990 issue does not work with my printer. In the October issue a gentleman wrote with a similar problem. I own the Commodore version of the STAR LC10 colour printer and it still doesn't work. Can you tell me how the DIN switches should be placed? In my opinion the printer gets the command to give a 'return' after every row. In each row the printer should have to print each colour twice before a 'return' is given instead of printing each colour on a new line. As a user of the C128D I naturally also do some programming in BASIC. As an amateur in programming I have a few problems. If possible, could you tell me how I can do the following: show the directory of a disk on the screen in the way that I want, check if the right disk is in the disk drive, check if the right program is on the disk, save a program whose name consists of two variables (eg. "Rekenigen.90" consists of "Rekenigen" and "90"), load a program with these two variables and how to make a hardcopy of any screen shown using the function keys.
Ruud Duits, The Netherlands.

Dear Ruud,
With regards to "Picture Print" it works with the DIN switches set to the positions that they were in when the printer came from the manufacturers - in other words, all switches in the ON position. The reason the printer is not doing what you want with the carriage returns is likely to be the fault of the positioning of the release lever, situated at the rear on the right. You must use perforated paper and pull the lever forward for the program to operate correctly. Regarding your queries, you will find a comprehensive routine for displaying the directory of a disk in the Database 7B program published in CDU a short while back. This covers most things - one can also be found in the Directories Explained program in the February 1990 issue. You could easily convert these to 128 format. To check that the desired disk is in the drive, the easiest thing to do is to give the disks a unique ID when you format them. Then the program can use the drives direct access commands to read in the very first sector of the directory track. You then locate the necessary bytes with the B-P command and read them in. Check against what you know the ID should be and hey presto!! Have a look in your drive's manual to work out the exact procedure there. To check if a program is on a disk give the following commands: OPEN 15,8,15: OPEN 2,8,0,"programname": INPUT#15,E: CLOSE 2: CLOSE 15. If the variable E is not zero then an error has been generated. If this is the number for a file not found error then the right program is not on the disk. If the file is sequential, add ",S" after the program name and before the quotes. To load and save files consisting of two variables, possibly separated by a dot, simply enter LOAD A$+"."+B$,8 or for a sequential file OPEN 2,8,0,A$+"."+B$+",S" where A$ and B$ are the two variables. To make a hardcopy in the way that you want, have a routine that reads in a key from the keyboard and if it is a function key (use ASCII codes to check) jump to the routine. For the routine, it is best if you PEEK each of the locations in turn and convert the POKE code into an ASCII code by doing certain checks and adding or subtracting values. After every forty characters, give a carriage return. I hope that you will be able to sort everything out now.

PRINTER WON'T PRINT

Dear CDU,
I have just obtained a second hand Olivetti printer model PR15 (no manual). I connected this via the user port to my C128D and powered up. I entered a little test program in C64 mode (OPEN 1,4: PRINT#1,"HELLO": CMD1,"HELLO": PRINT#1: CLOSE 1) but when I typed RUN there was no printer response. Is this printer incompatible with the 128 or am I entering the data incorrectly? The printer powers up, feeds paper and the LEDs light up! The print head also does a little movement from right to left as if ready for action. Please advise. One other query - is it possible to obtain 80 column mode on my 1701 monitor? I have followed my 128 manual instructions but I only obtain a blank screen and a frozen keyboard. As an avid reader of CDU, I value your column and your monthly tips although regrettably I am a very amateur computer addict and as yet have none of my own to offer. I do hope you will be able to help.
M.McGrail, Manchester.

Dear Mr.McGrail,
As this is not a standard Commodore printer I can unfortunately give little help. It may be that the printer actually has to be used as a different device number. Try entering what you have already, but substitute the fours with fives. If that does not work then I would suggest that possibly, yes, the printer is incompatible with the 128D. I would ask though if any of our other readers have the same type of Olivetti printer and know the secret then could they please come forward and send me the relevant information. To obtain an eighty column display you need a monitor that supports an RGB input. The 1701 does not have an RGB port and therefore you cannot connect it directly to the RGB port on the back of the 128D. However do not despair! A company called FSSL stock and item called a 40/80 column adaptor which will work with all monitors that have a composite video connection, which the 1701 does. This adaptor will allow an 80 column display from the 128D with your monitor. Their address for the purpose is FSSL, Masons Ryde, Defford Road, Pershore, Worcestershire, WR10 1AZ and their telephone number is 0386-553153. The product costs around twenty pounds. I hope that has helped a bit.

CONFIGURING A JACKET

Dear CDU,
I read with considerable interest Mike Gregory's Full Disk Jacket program on the Volume 4, Number 1 disk. Whilst I understand that Mike Gregory uses a Gemini-10X printer and he does say that the program may need to be changed, I decided to try it out and began to use it with great success in conjunction with my STAR LC10C printer. Great success that was, until I tried to produce a jacket for a double sided disk. In this instance the reverse side of the jacket was printed not upside down but back to front. The output from the printer is in the correct place on the jacket but the letters are turned vertically (sample enclosed). Could you perhaps let me know if any changes are needed to the program for my printer and also what these changes are. I do not have a great deal of programming knowledge and would greatly appreciate any help which might be forthcoming.
Mike Hill, Lincolnshire.

Dear Mike,
The sample that you enclosed illustrates the point about printing the reverse side of the jacket for double sided disks. When your sample is folded everything is perfect except that the letters are the normal way up and not inverted to produce a correct printout. The secret to this lies with the DATA statements in the program that are used to define the upside-down character set. There are a number of other things in the program, related to the choosing of a new character set, that will need to be changed to suit the STAR LC10C. It is rather difficult for me to pinpoint these exactly although in the near future I hope to present you with the alterations to make the program run correctly on an LC10C. Perhaps in the meantime you could print two jackets for the one disk and stick them on either side. Sorry I can't be of more help at the moment.

TIP OF THE MONTH

First let me say that if you are waiting for a tip to be published, please be patient - we have a great number and I can only print one or two each month. Rest assured that unless I have already told you otherwise, your tips will be published at the first available opportunity. This month's tip was programmed by me after having received the original idea from a Mr.David Colter. It will prove very useful for machine code novices who are finding it tiresome to constantly be converting binary numbers to decimal and decimal numbers to hex and so on and so forth. The very short program, filed as TECHNO TIP will output a complete list of all the numbers from 0 to 255 inclusive to your printer in decimal, with the binary and hex equivalents given along side. I am sure that a great number of you will find this program useful. Thanks to David for the original idea.

Well, this has been a very short Tip of the Month because no explanation of the program is really required. So I shall finish this month's session off by saying that if you do have any programming problems or you have any tips that you would like published, or if you simply want to air your views, then please write to the "T-Team" at CDU TECHNO INFO, 11 Cook Close, Brownsover, Rugby, Warwickshire, CV21 1NG. Don't forget those packs of disks that are on offer. See you again next time.

2 FOR THE C128

Two simple utilities for C128 users Neville Duguid

Here, for all users of the C128 are two simple utilities that could prove to be very beneficial. The first is a RAMCHECK program (sadly lacking on the C128) and the other is a test for accurate SOUND reproduction.

RAMCHECK 128

Bad RAM can be a mental health hazard to the Commodore 128 and its users. Normally reliable programs can crash without warning. Nonsense error messages sometimes appear. Files may appear to corrupt spontaneously - even after your disk drive has come back from the repair shop. Given enough time, a single bad byte can demoralize a Commodore 128 user to the point where he believes his computer is only useful when emulating a C64 In one respect, he is right. The C64 checks its RAM. The C128 does not.

IN THE BEGINNING

On power-up, the 64 mode Kernal examines every address from 1024 upwards until it finds a byte that will not function correctly as RAM. As BASIC needs contiguous workspace, the C64 treats this bad byte as the end of usable RAM and incorporates the number of good bytes it has found into its start-up message. Usually the terminating "BAD BYTE" turns out to be the start of ROM or a cartridge, but even if it Is not, BASIC will never attempt to use that byte. Unless you check the actual "BYTES EREE" message, or later get an "OUT OF MEMORY" error, you may never even need to know it is there.

The 128 mode Kernal performs no similar check. BASIC is responsible for the seemingly equivalent "122365 BYTES FREE" message. This figure is copied direct to the screen from ROM - along with the rest of the start-up text.

Commodore 128 users should therefore occasionally CHECK YOUR COMPUTER'S RAM !

NOTHING TO IT

It is easy to check the "BUSINESS END" of Bank 0. With the power off, remove any cartridge or other device from the expansion port, then power up with the Commodore key depressed. If the 64 mode start-up message contains "38911 BYTES FREE", 64 mode programs using only RAM between addresses 1023 and 40960 should perform flawlessly. (C64 BASIC programs are normally in this category). If you can confirm that a 128 mode program uses only Bank 0 RAM in this address range, than it too should cause no problems. Unfortunately 128 mode BASIC programs do not qualify. The BASIC program text is stored in Bank 0, but the variables created when the program is RUN make widespread use of Bank 1. (Even a direct mode command like PRINT"HELLO", which seems to copy a literal string from one part of the screen to another, makes Intermediate use of Bank 1).

THERE IS AN ANSWER

All this technical Information has been included solely to make you think:

"There has to be an easier way." There is! Turn on your C128, insert this month's disk into the drive then;

BOOT"RAMCHECK 128"

(If your default disk drive is not a 1570 or 1571, and your C128 is one of the earlier models with Version 0 ROMs, you may need to use the alternative;

BLOAD"RAMCHECK 128":SYS2816

The program will report either 'RAM OK' or give a list of bad RAM addresses in HEXADECIMAL. It uses a check similar to the one performed by the C64 Kernal. Starting at address 1024, it switches every bit in every byte both on and off and checks the results prior to restoring the original value. Unlike the 64 Kernal, it goes right to the top of memory in both RAM banks, regardless of how many bad bytes it finds on the way. The test takes just over four seconds when the computer is in EAST mode - nine seconds in SLOW.

The program has been made as short as possible - just 2 blocks in your disk directory. This increases its chances of operating successfully if your computer really does have bad RAM - and also explains the lack of any "bells and whistles."

NO PROBLEMS

In spite of the programming shortcuts used to achieve its speed and brevity, "RAMCHECK 128" remains compatible with BASIC and the rest of the C128 system. The STOP key is available as usual, and you can even use the Jiffy Clock to time the program if you wish;

BLOAD"RAMCHECK 128" :if not already loaded

```
FAST:TI$="000000":SYS2816:PRINT
TI/60"SECONDS":SLOW
```

(80-column users may want to leave out 'SLOW' at the end - it is there to unblank the 40-column screen if required).
When you get a bad report in FAST mode, you should also try SLOW - the problem may not be "BAD" RAM so much as "SLOW" RAM. If you have a printer set up as device 4, and the list of bad addresses is too long to fit on your screen (or you would like to take a copy of the report to the repair-shop);

```
OPEN4,4:CMD4:SYS2816
```

If BASIC won't work at all, RAMCHECK 128 can be started from the MONITOR. Hold down the RUN/STOP key as you press the Reset button, then;

```
L"RAMCHECK 128",8
J 0B00
```

Should the program find bad RAM on your computer, take note of the addresses and run it again. Check whether the addresses reported are the same every time. This can be helpful to a technician assessing the nature of the fault.
In case you feel you can't place too much reliance on a program that does nothing more than print "RAM OK" every time you try it, here is a way to trick "RAMCHECK 128" into thinking it has discovered some harmless "BAD RAM".

```
X  (if still in the MONITOR)

BLOAD"RAMCHECK 128"
POKE 3048,3:SYS2816
```

This falsely informs the program that only three configuration registers need to be skipped. The preconfiguration registers which are present at $FF03 and $FF04 (in all banks) do not retain information written to them, so RAMCHECK 128 should report them as "BAD RAM" if it is performing its job correctly.
$FF05 also gets reported even though it has not been bad. (Meddling with the configuration registers causes the program to get momentarily confused about whether it is looking at RAM or ROM). If this is the only thing RAMCHECK 128 ever finds wrong with your computer's memory, you can still appreciate the benefit to your own peace of mind.

Output produced by "RAMCHECK 128" used on C128 with faulty RAM is as shown below;

```
This is the BASIC screen:

ready.

boot "ramcheck 128"
searching for 0:ramcheck 128
loading

testing
bank 0... ram ok

bank 1... 0f8d 1f8d 3f8d 4f8d 5f8d 6f8d 7f8d 8f8d 9f8d
bf8d cf8d df8d ef8d ff8d = bad ram !!
ready.

This is the MONITOR version:

monitor
   pc  sr ac xr yr sp
; fb000 00 00 00 00 f8

l"ramcheck 128",8
searching for ramcheck 128
loading
j 0b00
testing
bank 0... ram ok

bank 1... 0f8d 1f8d 3f8d 4f8d 5f8d 6f8d 7f8d 8f8d 9f8d
bf8d cf8d df8d ef8d ff8d = bad ram !!
```

COMMODORE 128 TUNE-UP

If you have one of the earlier models with Version 0 ROMs, your Commodore 128's PLAY command is probably about two-thirds of a tone out of tune - more if it is a NORTH AMERICAN model. The problem may already be fixed if any ROMs have been replaced since manufacture, or if you have a C128D with Revision 1 ROMs fitted as standard.

SIMPLICITY ITSELF

An easy way to check is to RUN "C128 TUNE-UP" from this month's disk. (Use the RUN "filename" syntax - do not DLOAD separately). It will either BOOT "PLAYPATCH.128" (which re-tunes the C128 until you turn the power off or reset), or advise you that your C128's PLAY command already works correctly without intervention.
Should the message BOOTING "PLAYPATCH.128" appear, you will likewise need to BOOT "PLAYPATCH.128" on future occasions if you want your C128 to PLAY at standard concert pitch. (Alternatively, you could incorporate the relevant BASIC commands from "C128 TUNE-UP" into programs that use the PLAY instruction).
For example, if you write music programs intended to RUN on C128s other than your own, you might include this line, which loads the patch only if necessary:

```
10BANK15:IFPEEK(32766)=0THEN
BLOAD"PLAYPATCH.128",U(PEEK(186)):SYS4960
```

You should of course include a copy of "PLAYPATCH.128" on the same disk as any program that might attempt to load it. Although this example may

CONTINUED ON PAGE 48

FURTHER ADVENTURES IN "C"

THE CONCLUDING EPISODE OF OUR SERIES ON "C" JOHN SIMPSON

This month I am going to describe the standard INPUT/OUTPUT library. This is a set of functions which have been designed to provide the programmer with a standard I/O system for "C" programs. Regardless of how critical the application may or may not be, users seldom find the need to circumvent them as the functions are very efficient. The functions are also meant to be portable because most "C" systems exist in a well defined and compatible form. I will outline the more commonly used functions here rather than describe them all.

THE STANDARD LIBRARY

Whenever you write a source file which refers to one of the Standard library functions then you must include the line;

```
#include <stdio.h>
```

This should be close to the beginning, and the use of angled brackets < and > will direct the compiler to search for a file which contains standard header information. The file defines various macros and variables used by the input/output library.

GETCHAR AND PUTCHAR

To read a single character at a time from the "standard input", most often from the keyboard, we use the statement, getchar(). This will return the next input character each time it is called (we have used this extensively throughout the series).
If you need to substitute a file for the keyboard then the convention is to use the < character. For example, if a program called myprog uses getchar(), then:

```
myprog < datafile
```

will cause myprog to read datafile instead of the keyboard.

Whenever input is being read, and the read encounters the end of the file, then getchar() will return an EOF. The EOF constant is -1, but you can change this to whatever you require

The opposite to getchar() is putchar(). For example;

```
putchar(c)
```

will put the character c onto the default device, usually the terminal. The output can also be directed to a file by using the > character. If myprog uses putchar() then:

```
myprog > datafile
```

will write the output into the file called datafile
printf() also finds it's way to the standard output, and calls to putchar() and printf() can be easily interleaved.

```
/* an example routine which converts input to lowercase
*/

#include <stdio.h>
main()
{
int c;
    while ((c=getchar())!=EOF)
        putchar(isupper(c)?tolower(c):c);
}
```

Here we see two macros at work, these are defined in stdio.h, and test whether the argument is an upper case letter - isupper. If it is, then it returns a non-zero (true), and if not then a zero (false). The macro, tolower, will conveniently convert the upper case letter to a lowercase.

FORMATTED OUTPUT

There are two routines, printf(), which is used for output, (we have used also used this extensively throughout the series), and scanf() for input, which permit translation to and from character representation of numerical quantities. They also allow for formatted lines.

formally: PRINTF(CONTROL,ARG1,ARG2, ...)

printf() will convert, format, and print its arguments using control. The control string contains ordinary characters which are copied to the output stream, and conversion specifications. Each conversion specification is introduced using the percentage character, %.

Between the % and the character of the conversion there

may be:

- (minus sign) = left adjustment
a digit string = the converted number will be printed in a field at least this wide - if the argument has fewer characters than the field, then it will be padded with blanks.
. (a period) = separates the field width from the next digit string.
a digit string = the specific maximum number of (the precision) characters to be printed from a string
or number of digits to be printed to the right of the decimal point of a float or double.
l (letter ell) = a length modifier to indicate the data item is a long rather than an int.

THE CONVERSION CHARACTERS ARE

d = decimal notation
o = unsigned octal notation (without a leading 0)
x = unsigned hexadecimal (without a leading 0x)
u = unsigned decimal notation
c = single character
s = character string
e = float or double and converted to decimal notation in the form [-]m.nnnnnnE[±]xx where the length of the string of n's is specific to the precision.
f = float or double converted to decimal notation in the form [-]mmm.nnnnnn
g = use %e or %f, whichever is shorter; non-significant zeros are not printed.

If the character following the % is not a conversion character, then the character will be printed; therefore the % character may be printed by using %%

FORMATTED INPUT

The function scanf() is the input analog of printf() and provides many of the same conversion facilities, however, in the opposite direction.

Formally, SCANF(CONTROL,ARG1,ARG2, ...)

This reads characters from the standard input, interprets format specified in control, and then stores the results in arguments. The control string can contain conversion specifications to interpret input sequences.

blanks, tabs, or newlines (commonly referred to as "white space"), these are ignored.
ordinary characters (except %)
conversion specifications, %, * (an optional assignment suppression character), a number specifying a field width, and a conversion character

A conversion specification will direct the next input field conversion. Usually the result is placed in a variable pointed to by the corresponding argument. If a * is used the input field is skipped with no assignment made. An input field is defined as a string of non-white space characters which will extend to the next white space, or until the field width is finished. Since newlines are white space, then scanf() will read across line boundaries to find input.

The conversion character indicates the interpretation of the input field, and the corresponding argument must be a pointer.

CONVERSIONS

d = decimal integer
o = octal integer
x = hexadecimal integer
h = short integer
c = single character
s = character string - the pointer should point to a character array large enough to accept the string plus a terminating
0
f = floating point number

The conversion characters d, o, and x may be preceded by l (ell) to indicate that a pointer to long rather than int appears in the argument list.

Here's a scanf() example:

```
int i;
float x,
char name[30];
  scanf(%d %f %s",&i,&x, name);
```

with an input line of:
50 75.34E-1 George

will assign 50 to i, 7.534 to x, and the string "George" terminated with
0, to name. The three fields can be separated by as many blanks, tabs and newlines as desired.

A further example:

```
int i;
float x,
char name[30];

  scanf("%2d %f %*d %2s",&i,&x, name);
```

If the input line was:

98765 0123 35z67

then 98 will be assigned to i, 765.0 to x, skip over 0123, and place the string "35" in name. The next call to input will start searching from the letter z.

In both these examples, name, is a pointer and therefore must not be preceded by a &.

Here is a rudimentary input conversion function to demonstrate scanf():

```
#include <stdio.h>
main()
{
  double sum,v
  sum = 0
    while (scanf("%lf",&v) != EOF)
    printf("
t%.2f
n", sum += v);
}
```

When scanf has exhausted its control string, or when input no longer matches the control specification then it stops. The value it will return is the number of successfully matched, assigned input items.

Finally, the arguments to scanf() must be pointers Probably the most common error is writing:

```
scanf("%d",n);
```

which should be

```
scanf("%d",&n);
```

THE SIBLINGS

Both functions, scanf() and printf() have siblings which are sscanf() and sprintf(). They both act in the same manner as their big brothers, i.e both perform the corresponding conversions, however, the siblings will operate on a string rather than a file.

Formally SPRINTF(STRING,CONTROL,ARG1,ARG2, ...)
SSCANF(STRING,CONTROL,ARG1,ARG2, ...)

sprintf() will format the arguments just the same as before, however now it will place the result in a string rather than the standard output. N.B. Make certain the string is big enough to contain the result.

sscanf() does the reverse It scans the string according to the format in control and finally places the resulting values into arg1, arg2 etc. The arguments must be pointers.

ACCESSING FILES

Before a file can be read or written it has to be opened by the standard library function fopen(). This function opens a disk file for reading or writing. The string filename contains the name of the file, and the first character of the string mode specifies a 'r' read, or a 'w' write. The declaration required is.

```
EILE *fopen(), *fp
```

Here fp is a pointer to a file, and fopen returns a pointer to a file. FILE is a type name, like int, and not a structure tag. The actual call to fopen() a file in a program is:

```
fp = fopen(name, mode);
```

If there is any error when attempting to open a file, such as the file does not exist, then fopen() will return a null value. To read or write the file once opened we can use getc() or putc(). Here getc() will return the next character from a file, it will require the file pointer to tell it what file it is dealing with.

```
c = getc(fp)
```

This will place in c the next character from the file, and EOF is returned when it reaches the end of the file.

The inverse of getc() is putc():

```
putc(c, fp)
```

will place a character c on the file fp.

When we start a program three files are opened automatically with file pointers provided. These are 'standard input', 'standard output', and the 'standard error output' files. The corresponding file pointers are, stdin, stdout, and stderr.

getchar() and putchar() can also be defined in terms of getc and putc, stdin and stdout.

```
#define getchar()  getc(stdin)
#define putchar()  putc(stdout)
```

The function fscanf() and fprintf() can be used for formatted input/output files. The are the same as scanf() and printf() except the first argument is a file pointer which specifies the file to be read and the control string is the second argument.

Now we shall look at a small program which is used to concatenate files. Should there be any command line arguments they will be processed in order otherwise the standard input is processed.

```
#include <stdio.h> /* Concatenate by
Kernigman/Ritchie*/
main(argc, argv)
int argc;
char *argv[];
{
  FILE *fp, *fopen();
  if (argc == 1) /* number of args)
    filecopy(stdin);
  else
    while (—argc > 0)
      if ((fp = fopen(*++argv, "r")) == NULL) {
        fprintf(stderr"CAN'T OPEN %s
n", *argv);
        exit(1);
```

```
        }
        else {
          filecopy(fp);
          fclose(fp);
        }
    exit(0);
}

filecopy(fp)  /* this copies the file to standard output */
FILE *fp;
{
  int c;
  while ((c = getc(fp)) != EOF)
    putc(c, stdout);
}
```

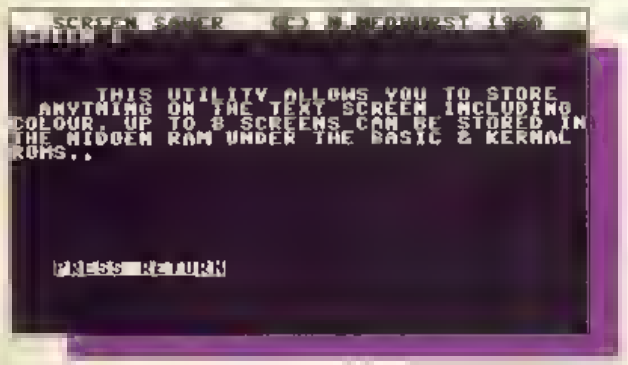
File pointers stdin and stdout are pre-defined in the i/o library. They are constants and may be used anywhere an object of type FILE * can be.
The function fclose() is the inverse of fopen() and breaks all connections between the file pointer and the external name established at fopen(), thus freeing the pointer for another file, and clearing the buffer in which output from putc() was placed. When a program terminates normally, then for each open file, fclose() is called automatically.
The program signals errors in two ways. The diagnostic output of fprintf() will go to stderr which directs it to the user terminal instead of a file. The program also uses the standard library function, exit(). A return value on exit of zero signifies all is well, and a non-zero value signals errors

MACROS FOR TESTS AND CONVERSIONS

isalpha(c) - non-zero if c is alphabetic, zero if not
isupper(c) - non-zero if c is upper case, zero if not
islower(c) - non-zero if c is lower case, zero if not
isdigit(c) - non-zero if c is digit, zero if not
isspace(c) - non-zero if c is white space, zero if not
toupper(c) - converts c to upper case
tolower(c) - converts c to lower case

TO CONCLUDE

Well, this brings our little adventure to a close. I hope that the foregoing episodes have helped you to enter the world of computer programming using the "C" language. There are many useful as well as excellent books on the market which can take you much further into the language than can our brief encounter. It does, to the newcomer, seem a daunting language, but with effort and practice it can, like anything else, be overcome until proficiency is a byword, and you can call yourself a "C" specialist. Once learned, with small, machine dependant, differences, one can then program on almost any machine which supports a "C" compiler, from Amiga to IBM.



SCREEN STORE 64

Store and recall your screens including colour
M. MEDHURST

The purpose of screen store is to store and recall up to eight screens including colour. The screens are stored under the Basic ROM and the colour is stored under the KERNAL ROM. Screen store can be used in PROGRAM mode, DIRECT mode, BASIC or MACHINE CODE. The code sits in memory at $C000 (49152) and is accessed with three SYS calls. Before storing any screens the SYS49158 call must be made to set the screen counter to zero, the screen counter is at memory location (2).

HOW IT WORKS

To store a screen just call up SYS49152 and the screen is stored. Each time a screen is stored the screen counter increases by one, if more than eight screens are stored a SCREEN MEMORY FULL error will be displayed and the program will stop. To recall a screen the call SYS49155 is made and the last screen to be saved will be displayed, if a SYS49155 call is made without any screens being stored a NO SCREEN error will be displayed. This is caused by the screen counter being at zero. The number of screens stored can be checked by PEEK(2) 1-8.
Having stored a number of screens they can be called up a random by POKEing any number from one to eight into location (2) and calling SYS49155 .The screens are normally stored on a last in first out basis.
If a screen is to be stored in any particular slot first POKE2,slot number 1-8 and then SYS49152 to store it.
A demo program is on disk and on running this it will load the code,
'BIN.SCREENSTORE'. This can he loaded on it's own by load'BIN.SCREENSTORE',8,1 followed by NEW to reset the basic pointers.
There isn't much else one can say about this easy to use program except give it a try

KANGAROO KORNER

The Third Segment

We conclude ELAINE FOSTER'S series on useful programming routines from Down Under

In this series, we have looked at ways of improving the usage of the ACTION REPLAY CARTRIDGE, and given you a few routines to aid you in your program development skills. The series concludes with two more routines to add to the ones already provided. These are SPACE INSERTER and LOADER ADDER. The first of these is, LOADER ADDER, which is a continuation of the article discussed in the DECEMBER issue of the magazine.

LOADER ADDER

Part 1 discovered that very long freezer-copied programs might not be compatible with tastloaders. This was overcome by adding your own loader on the back of the PRELOADER of the copied pair. It showed how to make a simple Machine Code (MC) loader to do this, and some other applications, e g. an autobooting loader (loads AND runs the larget program). Here, two further applications of this interesting technique will be described.

SYS NO MORE!

G. GORNU published a nice "Screen Saver" in YOUR COMMODORE, Dec 1989, p.53. It blanks the monitor screen after one minute, so saving its screen if you leave it on (which always happens). To run it, however, it was necessary to SYS53162 ($CFAA), and who is going to remember that horrid number, particularly if the documentation is not at hand? It is possible to remember the correct number if it is made part of the file name, e.g "Saver.53162", but there is another way: Load the MC program from Basic.

The simplest way to do this is:

LISTING-1, which not only loads the target MC program, but also supplies instructions every time you use it (or do you always have the original instructions at hand?):

```
10 IFA=0THENA=1:LOAD"SAVER.53162"
20SYSS3162:PRINT"[CLR][SCREENSAVER
INSTALLFD[DOWN]"
30 PRINT"SCREEN BLANKS AFTFR 1 MINUTE. PRESS
ANY KEY TO RESTORE."
40 END
```

The other way is to use a Machine Code Loader similar to the "SYS10037" routine described in Part 1. This is very fast and much more flexible. It is, in fact a hybrid, which makes everything much easier:

1) Enter the following Basic program, but type it EXACTLY as shown. Do not add or subtract a single byte, or it will not work.

LISTING-2

```
10SYS2154:PRINT"[CLR]SCREENSAVER
INSTALLED":PRINT"[DOWN]SCREFNBLANKS AFTER 1
MINUTE"
20 PRINT "[DOWN]PRESS ANY KFY TO RESTORF"
```

Of course the items in square brackets stand for single byte cursor control characters

2) Activate your Monitor, and enter the following items exactly in the Assembly mode. Do not include the semicolon comments (unless you use a proper symbolic assembler), which are only for interest:

LISTING-3

```
.> 086A LDA #$08      ;CHANNEL #
.> 086C LDX #$08      ;DEVICE #
.> 086E LDY #$01      ;SEC. ADDR.
.> 0870 JSR $FFBA     ;SETLFS
.> 0873 LDA #$0C      ;# CHARS IN FILENAME
.> 087S LDX #$84      ;LB OF NAME LOCATION
.> 0877 LDY #$08      ;HB OF NAME LOCATION
.> 0879 JSR $FF8D     ;SETNAM
.> 087C LDA #$00      ;0 FOR LOAD
.> 087E JSR $FFD5     ;LOAD
.> 0881 JMP $CFAA     ;JUMP TO 53162
```

and then the following in the "M" mode:

```
.:0884 53 43 52 45 45 4E 20 53 SCREEN S
.:088C 41 56 45 52 00 00 00 00 AVER....
```

Indeed, you may make the Basic introduction as long as you like, if you ensure that the BA of this routine (here $086A) is higher than the Basic EA, and that the SETNAM pointers (here $0876 & $0878) point to the beginning of the filename to load (here at $0884). And of course in the Basic line 10, the SYS must refer to the BA here ($086A = #2154).

The strength of this method, aside from being rather pretty, is that it allows each part to do what it can do best: Basic text is easy and takes little space, and Machine Code is fast and versatile.

3) Still with the monitor, enter:

.S"SS BOOT",08,0801,0890.

The result: a Basic file on disk, named "SS BOOT" which can be loaded and saved as a Basic program (eg, LOAD"SS BOOT",8). When it is run it automatically loads and runs "SCREEN SAVER" and enters SYS53162 automatically, so that you no longer have to remember it. If it is combined with an autorunning boot it only wants to be loaded, and SCREEN SAVER (or whatever target you choose) is loaded and activated immediately.

Create the above routine yourself to see how it feels, and then change it to load any other machine code program (See Part 1 for loading a Basic program).

A LOADER ADDER

Part 1 described how to make a loader adder, to ride piggyback on the PRELOADER of a long freezer-copied game. By tediously following that article you could make your own loader adder (like Listing-3) and insert it at the correct location on the PRELOADER. A friend who did not appreciate the beauty of MC only wanted a program to make the extra loader adder automatically. The result was ADDER.49152, also on the disk. It was designed to use on PRELOADERs made by the ACTION REPLAY CARTRIDGE, but could work with others by adjusting the location of JMP $FFD5; see the Opcodes under "Is No File Being Loaded??" in Part 1.

ADDER.49152 rather elegantly does all that Listing-3 does, and more. It automatically loads the target Preloader program which was the main point of Part 1, and then tacks on to the end of it a Listing-3 type loader (like "SYS10037"), readjusts the pointers appropriately and saves it to disk.

Remember that the main point of using the loader adder with the game PRELOADER was to make the game PRELOADER compatible with various Menu routines.
Technically speaking, the Flow Diagram for LOADERADDER shows approximately what is happening. The routine is located at $C000 (#49152) in high RAM because it needs to load the Preloader in the Basic area. It prints an option message and presents an INPUT for the usual reasons. It loads the Preloader, whose name you supply, sets up the various prefixes and pointers, and then transfers the Loader from high RAM down to Basic at the end of the Preloader. It then saves that combination to disk under a new name.

Incidentally, the difficulty of using fastloaders only applies to cartridge or RAM resident ones. ROM "Turbo" systems seem to be immune to such problems.

The advantage of the ADDERLOADDER program (which pokes the MC in from Basic) is that you can modify the code without using an Assembler. Merely follow the REMS. You may change the texts as you like, as long as the new one is not longer than the old, and if it is terminated by a zero. In line 670 you can change the "+" PRELOADER prefix to whatever you please as long as it is only one character; it is there only to make it different from the original. In line 590 you can change the requirement for a "1" prefix before the main target name, eg some freezers use a "-" instead. If you know what you are doing you can even modify lines 630-640. Use your monitor to find where JMP $FFD5 is located (or JSR $FFD5) in the PRELOADER and if it is not at $0835, use the right LB and HB of it in those lines; see the REMs.

The charm of this routine lies in the fact that the SAME set of bytes is used to load the Preloader, as is transferred down to the end of it, as it saves that combination to disk. Try to do that with Basic! It is one of the ways where "C64 thinking " saves space and promotes efficiency. When you have megabytes of room your programming tends to get very sloppy. Some of my megabyte friends are astonished that I can do ANYTHING with only 64K (INCLUDING the "System" in ROM), but my little "64" supports a nice Word Processor, a Symbolic Assembler, a mountain of Games and ...,.....))

SPACE INSERTER

Programs in Machine Code often use text requiring various amounts of space on the screen. At the end of a line this can be achieved with RETURN (byte 13), but within a line you will see long lines of $20 (dec 32), one byte per space.
This short program automatically inserts spaces (or anything else) in the output to screen or printer, and for the expense of only 54 bytes for the routine. It gives you any amount of spaces anywhere without taking up that room in your main program! It becomes practical for programs of reasonable length using much formatting.
This magical result is achieved by this tiny control system which scans any Machine Code text and prints it (until a terminating zero is encountered). If it finds a 1 however, it shunts to the space inserting subroutine, INSERTER, and prints the number of spaces shown by the next number. So, in the midst of text, a 1 15 would mean "insert 15 spaces here". As usual, on the disk is the Machine Code program and the Source Code program. Below, for those of you without an assembler, is the Source Code for the routine.

INSERTER SOURCE CODE

```
CHROUT      EQU $FFD2     ;65490
INDEX       EQU $2
COUNT       EQU $4
CHAR        EQU #32       ;SPACE
            ORG $080D     ;SEE TEXT
SET.INDEX   LDA #<TEXT    ;ELSEWHERE
```

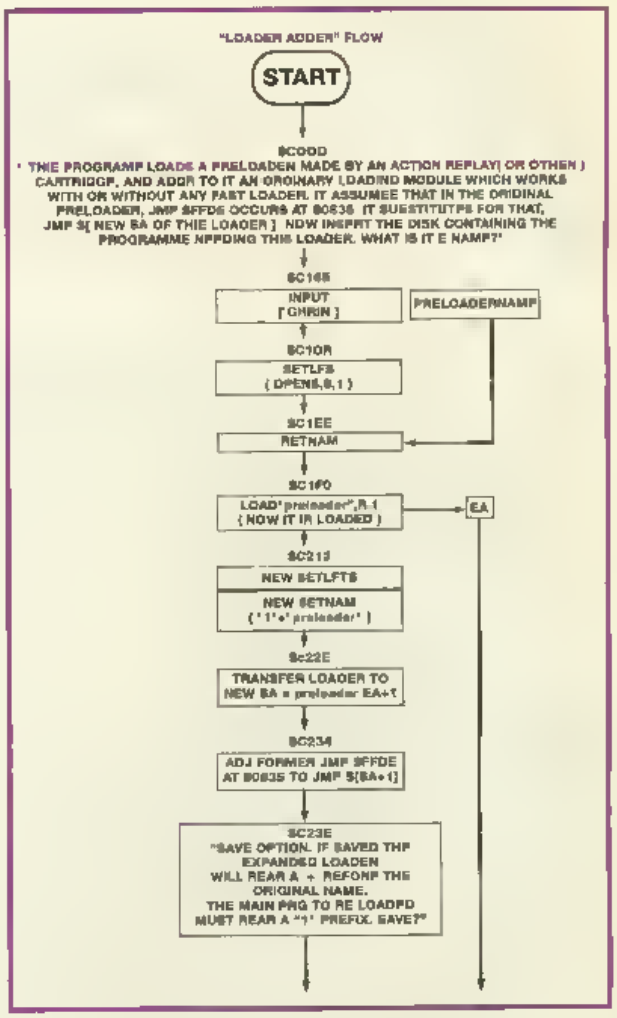
```
        LDX #>TEXT
        STA <INDEX    ;EOR ADDRESSING
        STX >INDEX
PRINT   LDY #0        ;THIS IS AN
LOOP.1 LDA (INDEX),Y ;ORDINARY PRINT
        BEQ END       ;ROUTINE, TO
        JSR CHROUT    ;THE SCREEN
        INY
        CMP #1        ;INSERT SPACE?
        BNE CONT.1    ;N.CONTINUE
        JSR INSERTER  ;Y=INSERT SPACE
CONT.1 CPY #0         ;END OE PAGE?
        BNE LOOP.1    ;N.CONTINUE
        INC >INDEX    ;UPDATE
        BNE LOOP.1    ;THE INDEX
END     RTS           ;RETURN TO MAIN PRG
;
INSERTER LDX #0       ;RESET COUNT
        LDA (INDEX),Y ;# OE REPEATS
        STA COUNT     ;STORE IT
        LDA #CHAR     ;LOAD SPACE
LOOP.2   JSR CHROUT   ;PRINT IT
        INX           ;NEXT COUNT
        CPX COUNT     ;DONE
        BEQ CONT.2    ;Y.EXIT
        BNE LOOP.2    ;N.PRINT ANOTHER
CONT.2 INY            ;NEXT BYTE
        RTS       ;RETURN FROM SUBROUTINE
```

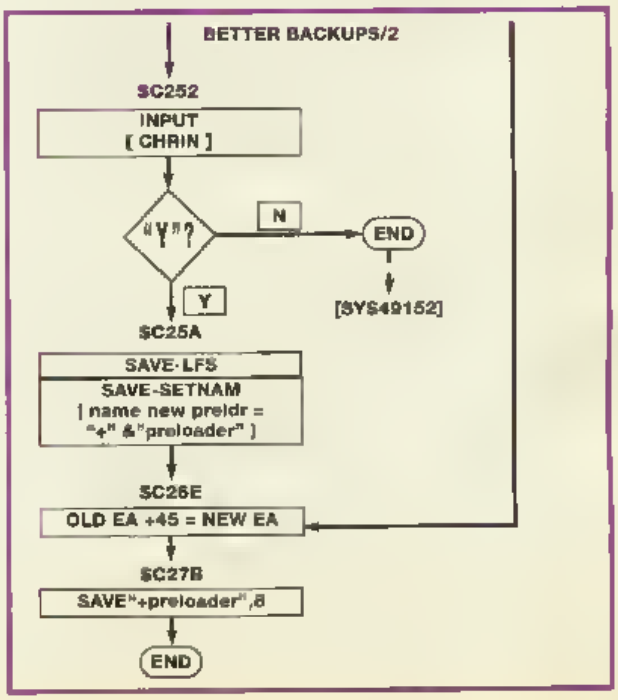
From the flow diagram you can see that in a normal text print routine (see PRINT), the print is scanned bytewise. If 0 is encountered it shows the end of text and it exits, else it prints the character to the screen. In this routine though, an additional jump is inserted: when 1 is encountered it shows that spaces are to be inserted, and it jumps to INSERTER. The number of spaces are recorded from the byte following the 1, and a counter prints them until they are finished. It then returns control to the main text PRINT routine.

The Machine Code beginning address for this example was chosen at 2061 so that it could be started from a Basic SYS2061, but of course it would be added to an already established program at any convenient location.

Load and Run the short program, SPACE INSERTER, which show how large blocks of spaces can be generated easily. If you have SPEEDY ASSEMBLER or any other compatible assembler, look at the Source Code for details.

The possibilities are endless, for example, you could fill the screen with reverse video spaces in different colours, or surround titles with various patterns. For the latter, you can see that Space is not the only character which can be inserted. The second byte at LOOP.2 controls the character. Using 65 in place of 32 would produce a series of "A's". A byte can be poked into that location from anywhere in your program, so the character to be repeated can be changed at will. If you want to take a little more space, a separate INSERTER routine (only 20 bytes) could be used for each different character. So, 1 40 could mean "insert 40 spaces", while 2 10 could mean "insert 10 cursor downs". For a large program the saving in RAM could be considerable, and it opens a new field of generous screen formatting.





DESIGNING A ROLE PLAYING GAME

Magic in Computer RPG's by GORDON HAMLETT.

The last part of this series dealt with COMBAT and various ways of dealing with it. This month we look at MAGIC.

It's a great feeling. Your warriors are tired and wounded. The party is lost and looking for a safe place to camp. The dreaded message appears on screen. Combat. You try to run away but a mixed party of ogres and trolls have surprised you and there is no escape. You are outgunned and you know it. Wearily, the fighters ready their weapons and prepare to advance when a voice from the back of the party says 'stand aside'

The magic user, who has so far proved to be excess baggage on this adventure steps forward uttering strange words. Flames begin to crackle at his fingertips. There is a loud whooshing noise. You look towards the enemy but there is nothing apart from a few blackened lumps and the smell of burning flesh. Your first fireball spell. It is a memorable experience and the fighters know instinctively that things will never be the same again.

LAYING THE FOUNDATIONS

Last month, I looked at designing a combat system and showed how it was the main part of a role playing system. Magic is the exciting part of the game but if anything, even greater care has to be taken over integrating a magic system. It is all too easy to get the balance wrong.

The actual mechanics of spellcasting are minor details and can wait until later in the article. The main problem is getting the amount of magic just right.

Traditionally, novice magicians are worse than useless. They can't fight and they know few spells. The fighters have a dual role to play. Not only do they have to kill the various monsters, they also have to protect any budding wizard who is incapable of looking after himself.

As the game enters its middle phase, so the balance is about even. The fighters are still important and the mages can play an important part too. For high level characters though, the situation is reversed. Now, magic dominates. Fighters become less effective. After all, they can only attack one creature at a time whereas the spell caster can target many. To go back to fireball example, the spell might have a 6x6 area of damage giving a potential 36 opponents. Quite a difference!

BALANCE IT RIGHT

The problem then is not to arrive at this 'final' stage of the game too quickly. It is worth mentioning here that all this applies to magic items too. There is a great temptation to reward players with magic rings of protection, wands of lightning bolts and swords of ultimate sharpness. They do make the scenario interesting but if the player is finding these and similar items at the start of the game, what are you going to give him as he progresses and you have to come up with bigger and better treasures.

A good example of a commercial game where the balance is totally wrong is Bard's Tale II. When I first reviewed it a couple of years ago, it looked great, even after many hours of playing. It was only when I got deeper into the game that I discovered all the above mentioned problems. Whereas a good game should encourage you finish, this had just the opposite effect and in the end, I just couldn't be bothered.

So how do you go about limiting the effects of magic? Well, there are several possibilities. You could limit the number of wizards in a party. Perhaps their psychic forces interact when they are too close together and reduce the potency of their spells.

CLASSES OF SPELLS

Another alternative would be to group spells together and then have a different class of magic user specialise in each type. For example, healing spells, sorcery spells, illusionary spells and so on. Bard's Tale uses this system to some extent and then spoils everything by letting characters change from one class to another.

The system used in SSI's Dungeons and Dragons series is somewhat different. A character can discover and study new spells by writing them in his spell book, but he or she can only remember a certain number of spells each day. Once these have been used up, the character has to rest for several hours in order to memorise a new set of spells.

Incidentally, it is worth while having a separate class that specialises in healing such as a cleric in D and D. Everybody is going to get wounded in battle at some stage so that this character really does become indispensible. If you have to a separate slot for a cleric, that is one less place in the party for a wizard. Casting

benign spells such as healing and protecting from evil does not have any great adverse effect on the game's balance. It is only the combat spells that are likely to cause you problems

The one feature common to all systems is that as a character becomes more experienced, so he can use a wider variety of spells. Similarly, the spells already known become more potent. For example, say your character knows a spell that inflicts an electrical shock onto an enemy. A first level character might cause 1-3 points of damage. You might increase this to 4-6 for a second level, 7-9 for a third etc but it is probably better just to multiply the damage by the level to get the damage range so that a second level mage causes 2-6, a third level 3-9 points of damage and so on. Again, this wider range dilutes the power of the spell caster.

MAGIC SYSTEMS

There are several different magical systems in current usage and it should be fairly easy to adapt one of these to your own needs. As already mentioned, in D and D, each magic user can learn a given number of spells at any one time. As soon as a spell is cast, the runes are wiped from his mind until he rests and relearns it. He can still learn the same spell twice if he wants to but must first learn it twice. This system works well enough but slows the game down when the party has to rest and sleep frequently.

Both Ultima and Bard's Tale employ a system of magical points it is then up to you to decide how you are going to 'spend' them. The more experienced you are, the more you have. A simple spell such as making a light might cost only one point whereas resurrecting a dead colleague might cost 50 - something totally beyond the means of a

neophyte mage.
One additional touch in the later Ultima games is that you have to buy or find reagents for the spells and mix them in the correct quantities. Four of the six reagents are readily available from your local herb shop but the others are rare and you have to solve many clues and face much danger before you find out where they are. This has the advantage of making sure that the powerful spells only become available when you are sufficiently advanced in the game to need them.
Even if you do not use this sort of system throughout, the idea of specific reagents for the most potent spells is a good one. You can control the game much more easily. It might even form part of the game itself. First you have to find and kill a black dragon to acquire one of his teeth, then you must locate and pay for the services of a great wizard who can mix the powdered tooth with some snake spit ...

SPELL USING

I have said very little about the nature of the spells themselves. This is because whatever system you use, the spells tend to include the same sorts of things - a small hit on one enemy, a small hit on a group of enemies, a medium strength hit on one enemy, the same on a group of enemies and so on. Of course, you can change the names, call them magic missiles, shocking grasps, fireballs, lightning bolts or whatever but they come down to the same thing in the end. Again, there is always a spell to make light, one to cure light wounds, cure serious wounds etc.
Think back to the role playing games you have played. They all boast dozens of different spells but how many of them do you actually use regularly? My guess is about 6-10. When you get caught up in a battle, do you cast a spell making the enemy's armour class weaker or giving one of your fighter's ogre strength? Of course you don't. You go in there and whack them. No shilly-shallying about.
One notable exception here is the Ultima system where the designer, Richard Garriott makes a point of ensuring that every spell is genuinely different and not just a variation or big brother of what has gone before. The moral here is simple. Don't just include something for the sake of improving a list. Make sure that it is an integral part of the game.

IT'S MAGICAL

Magic isn't just limited to spells though. One area of the game where you can let your imagination run riot is in the use of magical items. Weapons, rings, potions, wands, armour, items of clothing, packs of cards, statues, fountains etc. the list is endless. All you have to do is make sure that you don't overdo it and there are several ways round this.
Many of the items can have a limited number of charges. A wand of lightning bolts might only be able to be used three times before it crumbles and fades away. A potion might only have enough for one use and so on. You can also restrict who uses the item. The aforementioned wand can only be used by a wizard and if he is using that, he can't be casting one of his own spells at the same time. Similarly, if you have a magic two-handed sword, it is worthless unless you have a fighter trained in its use.
Other penalties could apply too. If someone wants to get the benefit of a magic ring, they might have to remove their gauntlets first, thus reducing their armour class and so on.Never give the player too easy a time of it. Remember too, intelligent monsters will use their magical items against you.

A FEW IDEAS

Not all magic should be good magic. Put in a few cursed items too. The trick here is to encourage the player to use an item straight away. Once a cursed item has been tried on, it cannot be removed until the appropriate spell has been cast or you have paid someone with the appropriate spells Curses shouldn't be too bad, more of an inconvenience. Drinking a potion might induce a temporary plague of boils on a character rendering him so ugly that no-one will talk to him. So, for a few hours, the party will be unable to trade with merchants or parlay with monsters - a fight becomes inevitable. A sword might give +2 bonuses but is extremely bloodthirsty and will not allow the wielder to run away from battle whilst an enemy is still standing.
One of the problems with magical items is discovering exactly what they do. It is pointless having a potion of dragon control if the player, not knowing what it does, drinks it all when surrounded by trolls Nor is it fair to say 'you have found a potion of dragon control'. Unless the container was labelled in some way, you are giving the player information for nothing. The solution is to make the players consult an expert who will charge very highly but will identify what an object does. Again, it comes down to giving with one hand and taking away with the other. Ok, so they now know what their potion does, but it has cost them all their hard earned gold to find out.

IN CONCLUSION

Another way of limiting the effects of magic is to give everything a percentage chance of working. A weapon might break or a spell fizzle and fail. If a character falls down a pit, potion containers might smash and wands snap. The original owner of an artefact might have conned. He thought he was buying a ring of protection whereas it was actually a ring of delusion or a brass curtain ring or whatever. These touches can bring humour and variety into a game and make the player just a little bit more wary when he does find something beneficial. Magic should be special, never commonplace. Gandalf was revered in Lord of the Rings and yet his magical powers were not that great. A fireball was about the limits of his power. If no-one else has the talents, even the smallest trick can seem miraculous.

THE ADVISOR

Intelligent decision making made for you BOB GARNER

This program is designed to help you to guess with more accuracy but it cannot hope to be the complete answer to all your decision making.

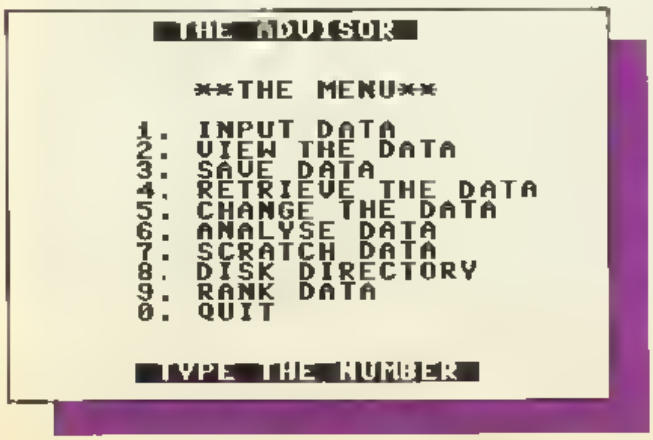
EXPERT SYSTEMS

Being a type of EXPERT SYSTEM. The ADVISOR requires that you choose what information is needed and to then set it in a logical order within the "SHELL". The program will give assistance but you should be aware of the "JARGON" associated with Expert Systems.

An Expert System provides a means to analyze information which is set against a number of norms These could be obtained from people recognised as "EXPERTS" in a particular field who decide on criteria to be used in mastering a skill or technique. Alternatively you can choose your own criteria for decision making to meet your needs. The ADVISOR shell allows setting of your own criteria using either of the two methods.

These criteria called ATTRIBUTES, belong to the subject under discussion. One example in relation to buying a book, could be "COST". Each attribute would then have a VALUE. Using the example of purchasing a book the value would mean "HOW MUCH?". It may be that a low cost of purchase is desirable. Although other ATTRIBUTES such as PRINT styles or the low measure of JARGON contained in the book, could outweigh COST.

By creating a set of ATTRIBUTES and their VALUES rules can be produced from which certain DECISIONS can be made. The complexity of decisions in purchasing is often the reason why so many turn out to be "BAD BUYS". The ADVISOR can over-ride the complexities to give as clear an answer as you wish.



SETTING UP THE ADVISOR

You can load THE ADVISOR from the CDU MENU or by typing LOAD"THE ADVISOR",8. When the program has loaded a short message appears on the screen, telling you that the dimensions are being set up. Electing to CONTINUE will get you a menu with 10 choices of action

** THE MENU **
1. INPUT DATA
2. VIEW DATA
3. SAVE DATA
4. RETRIEVE DATA
5. CHANGE DATA
6. ANALYSE DATA
7. SCRATCH DATA
8. DISK DIRECTORY
9. RANK DATA
10. QUIT

You type the number corresponding to your choice.

Let us look at each of the choices in turn to build up our own EXPERT SYSTEM. There are some already on the disk so you can skip the difficult part and follow the reasoning by typing 4 to RETRIEVE THE DATA, typing "BOOKS" at the prompt then RETURN. You can then VIEW THE DATA by selecting 2 from the MENU. If you are a hardy type of computer user we will press on.

STEP BY STEP GUIDE

Type the number 1 and you will be asked to NAME THE SYSTEM BEING CREATED so that it can be saved when it is completed Let us call the EXPERT SYSTEM - "BOOKS" and then press RETURN.
The ADVISOR then requests you to CREATE ATTRIBUTES. Type the ATTRIBUTES shown below in the square brackets, remembering to press RETURN after each.

ATTRIBUTE 1 will relate to [COST]
ATTRIBUTE 2 will relate to [PRINTING]

TRIVIA CHALLENGE RESULTS

We can finally reveal the winner of the TRIVIA CHALLENGE competition

CDU in conjunction with KEITH SUDDICK, the author of TRIVIA CHALLENGE, is pleased to announce the winner of our competition is;

MR .N. PRICE OF TEWKESBURY, GLOUCESTERSHIRE.

Mr Price obtained a score of 24,272 which was by far the highest that we received in the CDU editorial offices. Well done Mr. Price.

Because of the delay in marking this competition, consolation prizes of 25 blank CDU disks have been sent to the following people.

G.A.Cuthbertson; Glasgow
A.A.Robertson; Lancaster
Y.Dirke; Hamburg
C.Reading; Bristol
P.K.Price; Brighton
W.Schmidt; Bremerhaven
D.R.Gower; Bristol
S.Parker; Buckingham
E.R.T.Williamson; Lowestoft
F.Ashworth; Colwynbay
D.D.Baker; Belfast
P.Butterworth; Rochdale
J.K.Chamberlain; New Zealand
W.Dancer; London
D.Edmondson; Kettering
S.Sugars; Australia
E.Hallawell; Hull
I.Klause; Amsterdam
T.E.Marshall; Hereford
S.Mitchell; Sheffield
P.Perkins; Welwyn Garden City
Y.Soussi; Oslo
K.Keighley; Humberside
N.E.White; Swansea
M.Newing; Bognor Regis

We would like to thank everyone for taking the time and trouble to enter this fun challenge. Look out for more competitions in the near future.

top: A happy Mr. Price hears the news.

middle: An even happier Mr. Price receiving his prize.

bottom: An unhappy editor saying goodbye to the office monitor.

NOW IS THE TIME
TO CATCH UP ON ISSUES YOU HAVE MISSED

VOL 3 No.3 JAN '90
4 IN A ROW - Connect a row of counter.
FROGS IN SPACE - Leap to safety across the space lanes.
BLACKJACK - Don't loose your shirt.
LORD OF DARKNESS - Defeat the evil lord in true adventure style.
MARGO - Fly around and collect jewels and fuel.
JETRACE 2000 - Have you got what it takes to be best?
ULTIMATE FONT EDITOR - Create your own screens, layouts and characters.
SELECTIVE COLOUR RESTORER - Design your own system start up colours.
6510+ UNASSEMBLER - Transform 6510+ M/C into source
TRIVIA CHALLENGE - The first of 3 files for this superb game.

VOL 3 No.4 FEB '90
COLOUR PICTURE PRINT - Download your favourite colour screens.
BASE-ED2 - An update to our popular database system.
1ST MILLION - Play the market in this strategy game.
FM-DOS - Enhance your drives operating system.
GEOS FONTS - A further 4 fonts for Geos users.
HASHING IT - Relative file programming made easy.
MULTI-SPRITE - Make full use of up to 24 sprites.
DIRECTORIES EXPLAINED - Find your way round the directory jungle.
TRIVIA CHALLENGE - The 2nd part.

VOL 3 No.5 MAR '90
PLAGUE - Become your planets Guardian and Defender.
SURROUND - Reversi on the C64
GEOS FONTS - The last of 12 new Geos fonts.
SCREEN SLIDE - Create your own slideshows.
JOYSTICK TESTER - Put your stick(s) through the mill.
COLOUR MATCHER - Mastermind for the younger players.
SCREEN MANIPULATOR - Full use of the screen now obtainable.
VIDEO RECORD PLANNER - Keep tab on your home recordings.
TRIVIA CHALLENGE - The 3rd and final part of the game.

VOL 3 No.9 JUL '90
QUICK MERGE 64/128 - Another useful routine for your archives.
THE GAME PLAN - An aid to knowing whats where .
CHARACTER DESIGNER - Another designer for those without.
HASHBASE 128 - A powerful database for C128 users.
REVASM 64/128 - Two unassemblers for non Speedy Assembler owners.
SPEEDY UNASSEMBLER - An unassembler specific to Speedy Assembler users.
BANKS AND MEMORY - An aid to redefining screen and graphic memory.
GRAPHICS FACTORY - A novel way of getting in graphic design.
POT POURRI - A selection of useful routines for all users.

VOL 3 No.10 AUG '90
LIMBO 2 - The sequel to Limbo
SCREEN DESIGNER 128 - Screen designing made easy.
DATABASE7B - A database full of features.
LETTER MAKER - Text Screens made decidedly pleasing.
FUNCTIONS - Make full use of those function keys.
GAMES LIST CREATOR - Keep tabs on your games disks.
DUAL DISKCOPY - At last an intelligent disk copy program.
SEQUENCER64 - Musicians have a field day.
SECURITY - Put all those broken joysticks to good use.
SUPERBOOT! - Auto load your programs.

VOL 3 No.11 SEP '90
BANKING 128 - A simple way of keeping your money straight.
DISK DRIVER V4 - A simple disk utility.
AUTOBOOT 128 - C128 users get easy access to CDU programs.
READING BETWEEN THE LINES - Build your own adventure parser.
I.D.O.S. - A comprehensive drive utility.
PRICE CALCULATOR - Keep tabs on inflation.
B.O.S.S. - Yet another alternative to the standard Basic.
SCREEN DESIGNER/COMPILER - Impressive screen layouts for all.
LANDSCAPE ROUTINE - Beginners guide to scrolling backdrops.
SAMPLE KIT 64 - More sampling for all you musicians.

VOL 3 No.6 APR '90
BAR PROMPTS - M/C input routine.
HI-LITE BARS - Input routine but in Basic.
TEXAS DEMO - Example of using Basic in demos.
CHARS TO SPRITES - Convert UDG's to sprites.
FONT FACTORY - Complimentary program to the above.
3D-TEXT MACHINE - Impressive 3D text screens the easy way.
SCREEN ENHANCER - Makes full use of the screen easy to achieve.
SPREADSHEET 64 - An excellent, easy to use spreadsheet.
MINI-AID - 3 short utilities to aid the Basic programmer.
C128 COLLECTION - 3 very useful C128 programs.

VOL 3 No.7 MAY '90
NUDGE - FLD explained in laymans terms.
WINDOW WIPER - An alternative screen wipe system.
CHARACTER EXTRACTOR - Borrow those nice character sets you see.
MAZE GENERATOR - Create your own fun.
HIRES ANIMATOR - This difficult subject made easier.
SPRITE DRIVER - Platform game designing without the fuss.
ROTOTRON - Demonstration of Sprites and Sound.
TEXT COMPRESSION - How to squeeze a gallon into a pint.
SCREENS - Make up your own help screens and keep them in memory.
INTERRUPT POINTERS - Geos style windows and pointers for you.

VOL 3 No.8 JUN '90
ALEATORY MUSIC - An alternative music system.
SPRITE BASIC - Efficient sprite handling through Basic.
SPRITE GENERATOR - Another sprite editor for your library.
MUNCHER - Pacman returns with a vengeance.
ASTRODUS - Escape the spaceship Astrodus in this adventure.
1581 DIRECT ACCESS - Find your way around the 1581 disk drive.
PERSONAL ORGANISER - Design your own organiser pages.
128 CONVERTOR and MATHS AID - 2 more for C128 users.

VOL 3 No.12 OCT '90
ROLL'EM - An example of Graphics Factory in use.
COLOUR MATCH - A short utility for C128 users.
SPREAD-ED - The 3rd in the 'ED' series.
RASTER EDITOR - Put those raster lessons to good use.
ADDRESS BOOK - An unusual address base.
SUPERSORT64/128 - Sorts have never been easier.
SPRITE EDITOR C128 - Another utility for 128 users.
GRAPH-ED - The last in the 'ED' series.
BACKGAMMON - The popular board game gets an airing.

VOL 4 No.1 NOV '90
WAR AT SEA - C128 version of battleships.
FULL DISK JACKET - Keep track of your disks.
NEAGOX - Blast everything that moves.
NUMDEF - A basic game to test the reflexes.
MEMORY SCANNER - Look through memory the easy way.
MONEY 64 - Budget planning for the 90's.
XINOUT - An alternative input routine.
CALENDAR C128 - No more buying of calendars.
GOMOKU - An interesting variation of 'GO'.
SMOOTH SCROLL DEMO - Create your own scrolls.

VOL 4 No.2 DEC '90

tLS (German Program) - A C128 language tutorial.

SCREEN DESIGN CORRECTION - An update to this excellent utility.

BETTER BACKUPS - Help for ARC users.

MACHINE CODE GEMS - A suite of MC routines to aid you.

COLOUR TABLE EDITOR - Get those raster bars right.

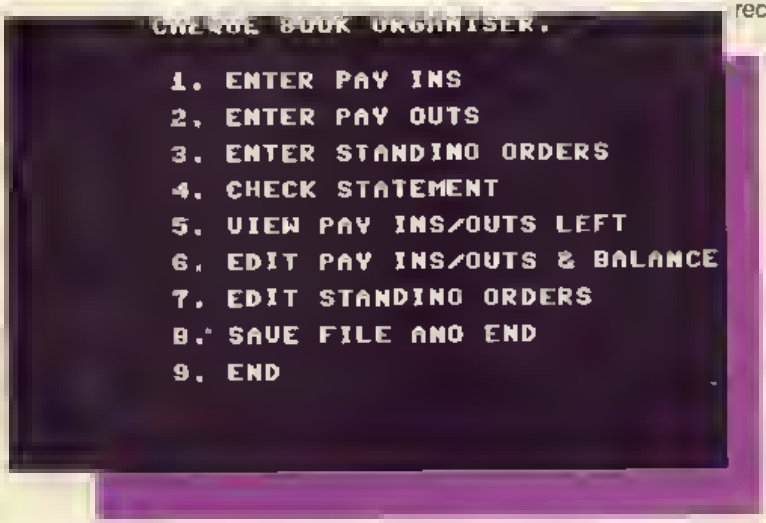
MULTITASKING C128 - An implementation for the C128.

Back issues of CDU are available at £3.25 per issue, which includes postage and packing. All orders should be sent to:- Select Subscriptions Ltd, 5, River Park Estate, Berkhamsted, Herts, HP4 1HL. Please allow 28 days for delivery.

CHEQUE BOOK ORGANISER

Take care of the pennies, the pounds take care of themselves PETER WEIGHILL

Cheque Book Organiser (CBO for short) is a program which will help you to check your bank statement at the end of each month. It can also help you to keep track of how much money you have left in your account, and is useful where you operate an interest bearing current account with your bank. They pay you interest if you have money in the account but charge the earth if you are overdrawn. CBO helps you avoid these charges. CBO is simple to use and easy to understand.



ABOUT CBO

When CBO is run for the first time, it will search for "CHEQUE DATA" and when not found, it will ask you to set up the balance (as shown on your last statement). Balance field size is upto #99999.99. It also asks you if the balance is CREDIT or DEBIT. To check, it asks you if this is okay, if NO then it repeats the two questions, if YES it goes to the main menu and gives you 9 options.

1. ENTER PAY INS

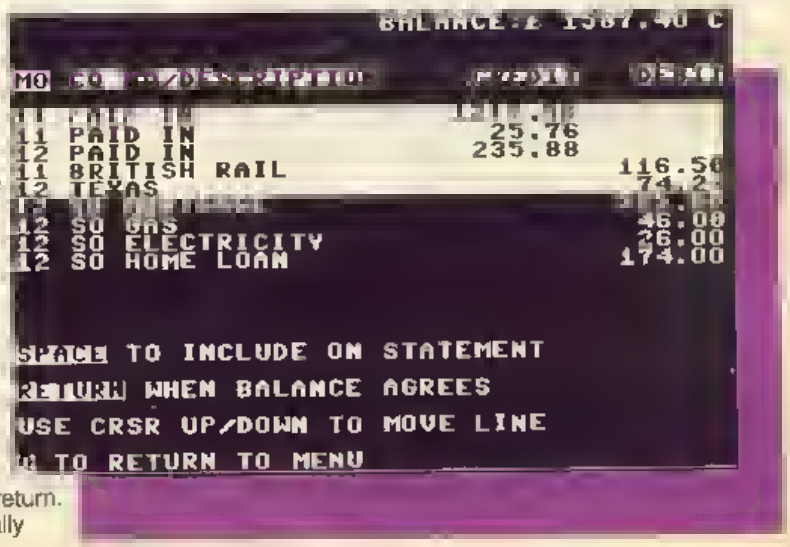
This option allows you to record any transaction where you have deposited money. You enter the amount and press return. You then enter the date (CBO automatically enters the backslash between day/month/year). CBO will show you what you have entered and will ask if correct. If NO you will have to input the information again. When you have input all your PAY INS press return for amount

2. ENTER PAY OUTS

Same as PAY INS except also asks for a narrative to record Cheque No and/or payee if you so desire. (Narrative up to 19 characters).

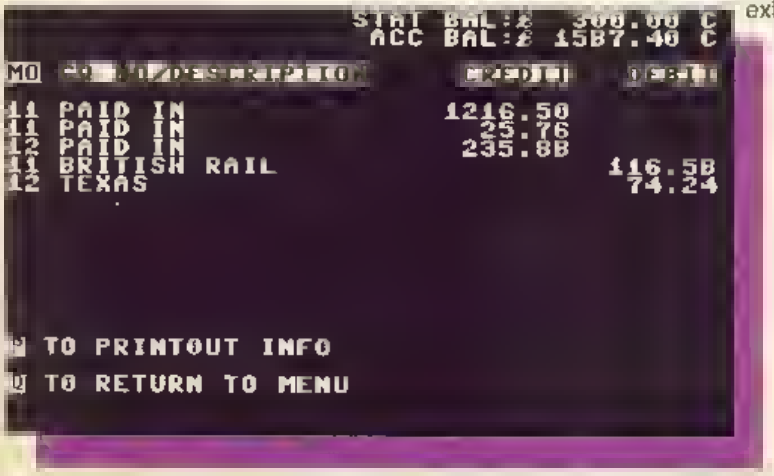
3. ENTER STANDING ORDERS

(You must set up Standing Orders first using option 7).
Asks "ARE YOU SURE". 'N' returns you to the menu, 'Y' asks for date. After date input, you are then asked if date is correct. 'N' will ask you to re-enter the date. 'Y' posts ALL Standing Orders to bank records and returns to menu.



4. CHECK STATEMENT

This lists last statement balance and cheques/receipts. (All transactions should have been entered using options 1-3 before you use this option).
The program allows you to check off those payments/receipts which appear on the statement Remember to enter any interest as a Pay In before using option 4.
It you check them off in the order that they appear on the Bank Statement, they will be printed out after you press return (it desired) in the same order as the statement
You use the space bar to select (or deselect) the items, and return when all items have been checked and the balance agrees.
The screen will ask "Are you sure?", and "Do you require a printed statement?". It will also tell you how many items are left to be processed by the bank



5. VIEW PAY INS/OUTS LEFT

This option allows you to view what cheques have been entered and/or were outstanding since your last Bank Statement It also shows you the amount ol money you have to cover any cheques you wish to write Press "P" tor a print out

6. EDIT PAY INS/OUTS AND BALANCE

This shows the last Statement balance and allows you to alter any entries previously made which were incorrect. Move the cursor to the required line and press return.
You will then be given a choice of DELETE, EDIT or QUIT (return to option 6) Edit offers you the choice of editing DATA DESCRIPTION, AMOUNT, CREDIT/DEBIT DESIGNATION. Once edit actioned, press "O" to return to option 6.

7. EDIT STANDING ORDERS

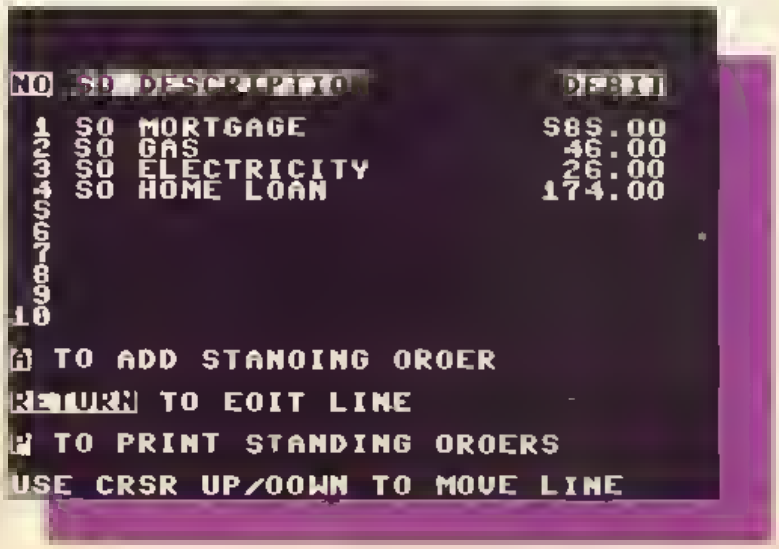
There is provision tor 10 MONTHLY Standing Orders. When you action option 3, ALL the Standing Orders are entered as due tor payment.
It any ot them are not required, they can be either removed prior to using option 3, or Edited using option 6. "A" allows you to enter a Standing Order. It asks tor amount and description (up to 16 characters), and automatically adds prefix SO to the description. It you do not use Standing Orders, but preter Direct Debits, then alter line 2 In the program by removing the word REM from it, leaving the remainder ot the line intact, and then resave the program using program name "CHEQUE BOOK 2" The program will then operate as for Standing Orders, but all reterences will be to Direct Debits (DD) and not Standing Orders (SO). Edits to existing data tollow similar routines to that outlined in option 6 Option to print out hardcopy ot Standing Orders or Direct Debits is available

8. SAVE FILE AND END

This option asks for confirmation prior to actioning the save routine. Any changes made using the options above require you to use option 8 to save your ammendments

9. END

As option 8 except that it does not overwrite the original data disk That is to say, use this option it data is only to be viewed and not modified



CONTINUED EROM PAGE 31.

appear cryptic to the uninitiated, its function can usually be duplicated more simply by:

BOOT"PLAYPATCH.128" :REM if required

Unfortunately this command is not adequate In programs intended for widespread distribution, as it will not work correctly from some disk drives when used by some C128s. LIST "C128 TUNE-UP" for examples of code intended to work with all drives on all Commodore 128s. (If you want to experiment with "C128 TUNE-UP", you will need to set DEV=8 In line 30 before it will RUN from memory).

EARS TELL ALL

For the musicians among us who trust only their ears when it comes to tuning, try this:

BOOT"PLAYPATCH.128"
PLAY "A":PLAY OFF:PLAY "A"

If you hear the same note repeated, your Commodore 128 has at least the BASIC LOW Replacment ROM and will have no further use for this program. If you hear two different notes, you should get better acquainted with PLAYPATCH 128 as you will probably be needing it. 'PLAY OFF' is a special instruction to disable the patch. Its effect is the same as SYS5251, but is more convenient to use as you don't have to remember an address Having installed PLAYPATCH 128, why would anyone want to disable it? One reason Is to relInquish the memory it uses in order to make way for other programs. If you (or a program) use some of the patch's memory when it is still active, you risk crashing the computer.

WHERE ITS AT

When active, PLAYPATCH.128 occupies $1360-$173B and $03E4-$03EF In Bank 15, where "$" denotes hexadecimal. BASIC programs do not normally use this memory, but machine language (ML) programs favour it. The patch can be restarted by 'SYS4960' without reloading, provided the segment at $1360-$14C4 remains undisturbed in memory. So, if you are likely to use any of the above-quoted memory locations for other ML programs, don't forget to 'PLAY OFF' first, or you may later be forced to reset your computer. (RUN/STOP Restore will not fix it). Don't worry if 'PLAY OFF' sometimes causes a Syntax Error for no apparent reason. This merely indicates that PLAYPATCH.128 was not enabled at the time. (Perhaps you previously disabled it). "C128 TUNE-UP" contains an example of 'PLAY OFF' which won't break a BASIC program if the patch is not in use. If you can't remember - and can't tell by ear - whether you last switched your PLAY patch 'ON' or 'OFF', there is no harm using SY54960 more than once to ensure your music gets played correctly.

With PLAYPATCH.128 installed, your Commodore 128 now has the ability to harmonize with other fixed-tuned instruments of standard pitch. Although this is not always important, some pieces just don't sound right unless played in their familiar key.

BASICS of BASIC

A series of Basic tutorials designed to make the beginner an expert
JOHN SIMPSON

It seems, from the many letters and telephone calls we receive at the offices of CDU, and also from an early prognosis of our recent readers survey, that there are many of our readers who are not initiated into the practise and noble art of computer programming, but who would very much like to be.
As one of the aims of this magazine is to help increase peoples knowledge and awareness of computers from a deeper viewpoint rather than simply playing games, or using business orientated applications, we have decided to initiate an in-depth series of tutorials dealing with the subject, the language of BASIC.

WHATS COMING UP

Within each Issue we will set out lessons, both in text and with "on the disk" examples, for the CDU Student to peruse, study, practice and digest. The lessons will commence from the viewpoint of the CDU Student having no language or computing knowledge or skill whatsoever, and will gently, yet thoroughly, progress through to a point whereby the CDU Student will, with conviction and certainty, be in the enviable position of having the ability to create programs for the computer to undertake many, many tasks.
The tutorials will cover all aspects of Basic programming techniques from NUMBERS and VARIABLES, LOGICAL OPERATIONS, DATA CONVERSIONS, INPUT AND OUTPUT, BASIC KEYWORDS, GRAPHICS · including SPRITE MANIPULATION as well as colourful SCREEN DESIGN and creation - SOUND EFFECTS AND MUSIC, through to glimpses into the hardware of the C64
Once the CDU Student has completed the course then the he or she will have a fine and detailed understanding of the workings of the machine and will be ready to move on to even more complex aspects of computing, such as languages like C (the "In" language of computing), or machine code (the actual language of all computer).

I WOULD LIKE TO REMIND OUR READERS AND POTENTIAL CDU STUDENTS THAT THIS COURSE WILL COMMENCE EROM THE VIEWPOINT OF "NO KNOWLEDGE", AND SO, THEREFORE, IN NO WAY DO WE WISH TO OFFEND YOUR INTELLIGENCE. SO, IF YOU HAVE A BEGINNER'S GRASP OE BASIC THEN POSSIBLY THE EARLY STAGES OF THE COURSE MAY APPEAR SIMPLE TO YOU, BEAR WITH ME AND THOSE

WITH A LESSER UNDERSTANDING THAN YOURSELF. IT WILL SOON REACH YOUR LEVEL OF EXPERTISE!

THE ORIGINS OF BASIC

The word BASIC is an acronym for 'beginner's all-purpose symbolic instruction code'. A programming language which was developed in the mid-1960s. Basic, as originally conceived, was a very simple language that could be learned very quickly. It was this simplicity of Basic which made it a natural choice as a programming language for the early microcomputers, and from there it rapidly became an established language.
Since then it has become somewhat more complex with many more instructions, and yet it still manages to retain the elements of simplicity. This is encouraged by the fact that the language itself is conveniently close to English which does help to make it reasonably easy to understand and to follow. As an example of this here are a few lines of coded computer instructions in three different languages, each of which accomplishes the same end result

1. BASIC

```
10 LET X = 1
20 IF X = 10 THEN END
30 PRINT X
40 X = X + 1
50 GOTO 20
```

LINE 10 Here we have set X equal to 1
LINE 20 In this line we test if X equals 10. If it does then the program terminates, otherwise it will continue to line 30
LINE 30 This line will print onto the screen the current value of X
LINE 40 On this line the value of X is incremented by 1
LINE 50 And this line redirects the program back to Line 20 to test if X has reached the value of 10, and so on..

2. MACHINE CODE

```
10    LDA #1
20 TOP  CMP #10
30    BEQ SKIP
40    JSR $FFD2
50    CLC
60    ADC #1
70    JMP TOP
80 SKIP RTS
```

3. C

```
main( void )
{
 int x,
 for(x=1;x<10;)
   printf("%d
n",x++);
}
```

As you can see, the basic example is fairly straightforward and not too difficult to follow. The Machine Code example is a mystery to the uninitiated, and the C language example will certainly require a lot more explanation.
However, as I said earlier, once we have reached the end of the course, which starts in earnest next issue, then the magic of machine code and the completeness of C will be a much more easily achieved goal, and we aren't talking football!

PREPARING A DISK FOR THE SERIES

The first item you will require for this course is a newly formatted disk ready to save examples, and "test" programs. To do this take either a new disk, or an old one (the programs upon which you no longer require), and place it into your disk drive. Next type the following, exactly as is printed here

```
OPEN 15,8,15
```

Once you have typed this line, press the key to the right of the keyboard marked RETURN. Now type:

```
PRINT#15,"N0:BASIC PROGRAMS,TU"
```

And again press RETURN. The disk drive's red light will now come on and you will hear activity within the drive. After about two minutes the red drive light will blink off. Your disk is now formatted and ready for use. However, you will now need to reset the computer back to it's original situation, so now type:

```
CLOSE 15,8,15
```

And then press the RETURN key

ENTERING COMMANDS

There are many keywords (71 to be precise), which the computer will recognise as commands to instruct it to do something. Let us examine one in particular. This is the word PRINT. When the computer comes across this word it knows that any statement following PRINT must be output to the screen (it could be to a printer, but I will be dealing with that later in the series)

If you formatted a disk, as outlined earlier, then you would have already used three Command Words (or keywords), namely, OPEN, PRINT#, and CLOSE. However, the important point is that after typing in the keyword, followed by the statement (such as 15,8,15), you pressed the RETURN key. This key is used to tell the computer to ENTER the statement you have just typed into its memory (this is why sometimes you may come across the word ENTER instead of RETURN). It really is important, however, to remember to always press RETURN at the end of each line.

TO BE CONTINUED NEXT MONTH.

f/s/s/i
COMPUTER SOFTWARE

SUPER SNAPSHOT v5

Well, it may have taken a few years of hard work and five powerful versions each one breaking new ground, but Super Snapshot has become the best cartridge in the world The list below details the main features of Super Snapshot v5, If you need a little more persuasion look back to CDU issue 19, you'll be impressed
So take a look at the red box you've got plugged in, and if our specifications knock it for six or you don't own a cartridge then don't just sit there, buy Super Snapshot v5 today!

FEATURES:

- ► All features available at the press of a button
- ► Works with all 64 (c) and 128 (D) computers
- ► Compatible with 1700/1764/1750 REU'S
- ► Snap any memory resident program into 1 file
- ► Save 7× faster and load 15× faster on the 1541, 1571 & 1581 Speeds of up to 25× faster when using Turbo 25 – even faster than Replay
- ► Super DOS Wedge
- ► GAME MASTER menu with sprite killer, infinitive lives generator and joystick port swapper
- ► Programmable function Keys
- ► Sprite Monitor
- ► Exclusive Character Set Monitor
- ► Exclusive Sound Sample Monitor
- ► Exclusive Boot sector support
- ► 300/1200/2400 Terminal program (40/80 column)
- ► SUPER DISK SNAPSHOT – our new super nibbler
- ► SCREEN-COPY now loads or saves in more formats and dumps in COLOUR to STAR LC10C printers and in 16 grey scales
- ► Improved full featured M/L monitor that DOES NOT CORRUPT MEMORY Interrupt, examine and resume any running program
- ► Drive Mon
- ► BASIC PLUS with 15 new BASIC commands
- ► FILE MANAGEMENT SYSTEM – scratch, unscratch, rename or adjust skew Includes our 1 or 2 drive file copier with partition support for the 1581
- ► Fast disk copier 1 or 2 drives
- ► OUR FILE COPIER, DISK COPIERS and NIBBLER MAKE FULL USE OF THE REU'S
- ► Sequential file reader
- ► Utility disk
- ► Plus 150+ Kracker Jax parameters

ONLY £34.95

Prices include VAT & U K delivery Overseas orders send advertised price plus £2 50 for Airmail Please send cheque, Postal Order or Credit Card details VISA/ACCESS orders accepted by 'phone

Masons Ryde,
Defford Road,
Pershore, Worcs.
WR10 1AZ.
Tel: (0386) 553153
Technical Support
Tel: (0386) 553222

ANIMATION STATION

Unleash your creativity with Animation Station, a powerful utility for generating all kinds of graphics on your Commodore 64 or 128 Built-in, predrawn pictures give you a head start on your creations Automatic generation of circles, ovals, squares, boxes, straight lines, typography and other geometric shapes gives you all the tools of a self-contained electronic drafting room Combine type and graphics on the screen, draw in many colours, even connect your VCR to create titles and graphics for your home movies Screen dump to your printer Koala Compatible Package includes design touch pad, pencil and graphics software. Ideal for GEOS users

ONLY £59.95

VIDEO BYTE 3

Digitize video images from your VCR, laser disk B/W or colour camera, off the air or cable TV New version 3 0 software features full re-display with multi-capture mode, menu select printing expanded colourizing features, save to disk and much more. The hardware is no larger than an average cartridge which plugs into the User Port The menu driven capture software is easy to use and pictures stored to disk can be imported into most popular drawing packages including GEOS Prints in full colour to the Star LC10 when used with SuperSnapshot

ONLY £79.95

HOME VIDEO PRODUCER 64

Nothing can make home videos so special Add titles, text, and even brilliant graphics to your favourite home videos with ease and the help of the Home Video Producer You have the choice of 10 typefaces 75 large full-colour graphics and ready made segments, but the most appealing aspect of the Home Video Producer is the ease with which you can do all this.

ONLY £29.95

C64 SLIMLINE UPGRADE CASE

Make your older C64 look like a newer model! All you need is a screwdriver and about 15minutes to transfer your C64 insides to this new case Complete instructions included

ONLY £12.95

CARTRIDGE PORT EXTENDER CABLE

Are you cramped for space behind your computer? Is it hard to reach your cartridge port to plug-in or swap cartridges? This handy cable is the solution The Cartridge Port Extender Cable connects to the cartridge port in the back of the computer and lets you plug in your cartridge to its other end. Since the cable is flexible you can locate the cartridge up to 11" away for easier access Not for REU's

ONLY £19.95

GEOS APPLICATIONS

GEOSCAN ART

This special type of GEOS art has been created using The HandyScanner 64 Pictures are scanned at 400 dpi from magazines books and papers to create the first geoSCAN ART Collection entitled The British Countryside and is packed full with Eagles, Owls and Butterflies

ONLY £6.95

GEODIRECTORY

A comprehensive book listing all available GEOS programs Details for each program is given including version numbers The geoDirectory is divided into sections covering Paint, Write, Spell, File Calc, Chart, Terminal, Graphic, Music, Animation, Games and many more.

ONLY £6.95

GEOTRONIX

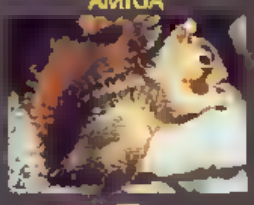
A professional PCB designer utilizing the GEOS environment Five double sided disks supply geoPublish with pre-designed components, sockets, edge connectors and layout grids in Photo Scraps Using the Photo Manager and geoPublish the circuit is designed and printed

ONLY £39.95

GEOS 64 V2 0	£29.95
GEOCALC 64	£24.95
GEOFILE 64	£24.95
GEOPROGRAMMER	£29.95
DeskPack Plus	£19.95
GEOCHART	£19.95
GEOS 128 V2 0	£39.95
GEOCALC 128	£29.95
GEOFILE 128	£29.95
GEOPUBLISH	£29.95
FontPack Plus	£19.95
Int Font Pack	£19.95

VIDEO COLOUR SPLITTER

AMIGA



ST

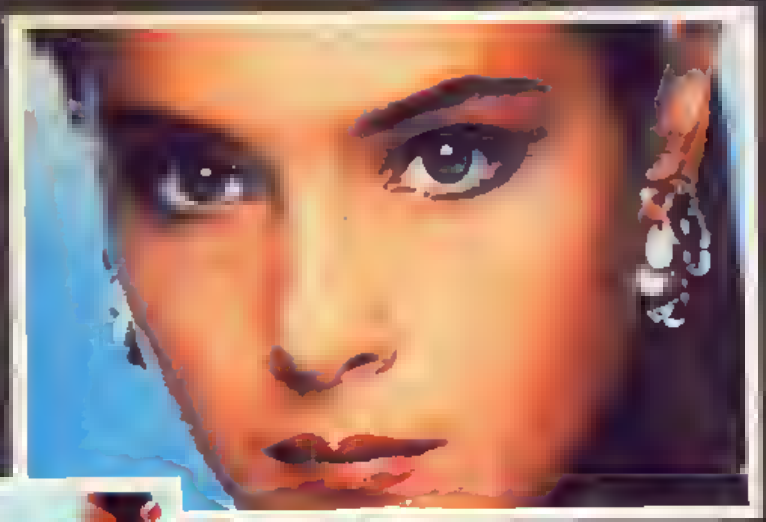
PC

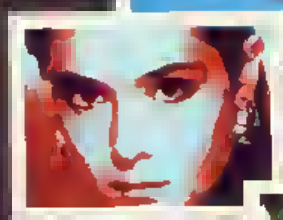
Actual Digitised colour screen - shots

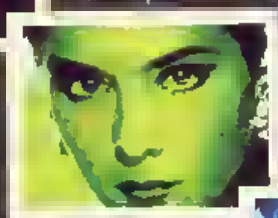
£69.95 Inc Vat

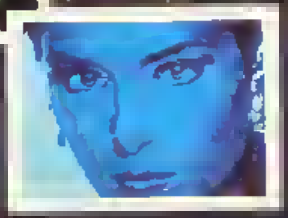
Vidi-RGB is an electronic filter which takes a colour video signal and separates it into the three primary colours (Red, Green and Blue) allowing each to be digitised.

Ideal for use with Vidi-Chrome & Frame Grabber or Digi-View Gold (By Newtek).









All these pictures are **actual unretouched screen-shots** illustrating the sequence of creating a full colour image using Vidi RGB, Vidi-Chrome and Vidi Amiga.

- For use with colour Digitisers replacing conventional Filter sets.
- Our Vidi - Chrome switches Vidi - RGB automatically grabbing full colour pictures in less than one second.
- Digitise full colour images direct from home VCR (must have perfect freeze frame)
- Digitise outstanding colour pictures direct from Canon's new Still Video Camera (an example shown on cover)
- Manual switching for maximum flexibility.
- Fully compatible with Digi - View Gold.

Limited

6 Fairbairn Road,
Kirkton North,
Livingston,
Scotland,
EH54 6TS.

Tel: 0506-414631
Fax: 0506-414634